ZX - РЕВЮ
N7,N8 1992

МКП "ИНФОРКОМ" 121019, Москва, Г-19, а/я 16

Письмо читателя.

" Привет, ИНФОРКОМ!

Всегда с удовольствием читаю "РЕВЮ" от первой до последней страницы, но простите, что-то в последнее время не пойму, что Вы затеяли. Вся страна объявляет доподписки на газеты и журналы и требует доплаты, а вы упорно стоите на своем. И слепому ясно, что на те девяносто рублей, которые стоила подписка в прошлом году Вам не продержаться. Вам там в Москве легко играть в благородненьких. Мы, мол не такие, как все. А вот нам здесь сермяжным что прикажете делать, когда все Ваше дело квакнет? Вы-то выкрутитесь, у вас ИБМ-ы всякие и прочие дела, а мне куда деваться с моим "Ленинградом", который я пол-года паял, если этих самых ИБМ во всем городе полторы штуки и никто не знает, как к ним подойти?

Знаете, ребята, давайте кончать это дело. Вы бы хоть с народом посоветовались. Может, людям и не жалко денег за "РЕВЮ". А вот если вы задумали это дело свернуть, как невыгодное, то имейте в виду, Родина Вам это не простит".

Вот такое пришло письмо. Мы не приводим имени автора, их несколько сотен, писем с таким содержанием. Мы выбрали не самое мягкое и не самое жесткое и сохранили авторский стиль. Такие письма приходят уже несколько месяцев. И привели мы его вовсе не для того, чтобы как это часто делают "по просьбе населения" отмочить какую-нибудь гадость. Ничего подобного. Никаких доподписок как не было, так и не будет. Мы твердо стоим на том, что если издатель принял от людей деньги авансом, когда червонцы были большими, то и отработать он их должен полностью. Это его трудности, если он не сумел или не успел их использовать по назначению своевременно - для закупки бумаги, техники, оплаты перспективных разработок.

Пусть доподписку просят те, кто все проел и промотал вместо того, чтобы организовать дело с перспективой.

Вместе с тем, это письмо явилось и поводом для глубоких размышлений. И мы сделали три генеральных вывода:

1. Наша работа нужна, ее ценят и если мы ее свернем, то Родина нам не простит.

2. Если наша работа нужна и мы ее не будем развивать, то она свернется сама и Родина нам не простит.

3. Если мы будем развивать свою работу так, как мы это могли бы и хотели, то нас загонит в гроб нынешнее положение с почтой и типографиями. Мы превратимся в некое подобие давно почившей организации "Книга-почтой", скончаемся и Родина нам не простит.

Замкнутый круг и разорвать его можно только если среди Вас, уважаемые читатели,

найдутся предприимчивые люди, способные подхватить наше дело и продвинуть в своих регионах.

Мы сосредоточиваем все усилия на подготовке новых материалов, а вопросы печати и распространения по регионам отдаем Вам на условиях лицензирования.

У нас лежат без движения еще десятки тысяч страниц необработанной информации и, если мы, как и сейчас, будем 90% сил и энергии тратить на почту и печать, то они так никогда до Вас и не дойдут.

Мы давно начали готовить развернутую серию книг по "Спектруму" и приглашаем включиться в дело их издания и распространения всех, кто имеет для этого физические и материальные возможности. Сейчас количество "синклеристов" в стране уже составляет несколько миллионов, а мы едва можем обслужить десятые доли процента. Только совместными усилиями мы дойдем до каждого.

Итак, назовем условно готовящуюся серию "Библиотека ИНФОРКОМа". Оговоримся сразу, что это будут книги совершенно новые, оригинальные, а не перепечатки того, что и так циркулирует по рынкам всей страны. Объем каждой книги - 200... 220 страниц. Что в эту серию войдет:

Самая ближайшая задача - сериал, посвященный графике "Спектрума". Пока он разрабатывается в четырех томах.

Т. 1 "Элементарная графика".

Он уже готов и за его издание и распространение можете приниматься хоть сейчас. В порядке презентации одна глава из этого тома напечатана в сегодняшнем номере "РЕВЮ".

Т. 2 - "Прикладная графика". Здесь рассмотрены вопросы деловой, научной, игровой, векторной, трехмерной, теневой графики и пр. и пр.

Т. 3 - "Динамическая графика".

Имеется в виду использование динамической графики для создания мультипликационных и анимационных эффектов в Ваших программах,

Т. 4 - "Дизайн Ваших программ". Не повторяя материалов первых трех томов, здесь рассмотрены вопросы применения специальных приемов и эффектов для получения полноценных программ. Эта книга является кульминацией всего того, о чем было сказано в первых трех томах. Один отрывок из этого готовящегося сейчас тома также представлен на страницах сегодняшнего выпуска.

Над томами 2,3,4 сейчас идет работа. По окончании работы над ними в перспективе планируется решить вопрос о целесообразности выпуска пятого тома "Справочника по машинной графике", но это уже будем смотреть по обстоятельствам. Здесь есть проблемы.

Во всяком случае, первые четыре тома сейчас готовятся в предположении, что никакого справочника нет и все, что нужно для понимания материала, в них имеется.

Кроме графического сериала, идет подготовка и других. Во всяком случае, для первых 15-17 томов материал уже собран и они находятся в работе. В потенциале если все пойдет хорошо, можно будет довести общее количество книг до 22-25.

Вторым крупным готовящимся сериалом будут тома, посвященные вопросам самостоятельной разработки программ - обучающих и прикладных, а также адвентюрных и аркадных игр.

Третий сериал чисто игровой, содержит готовые программы для самостоятельного набора. Отрывок из книги, посвященной настольным играм, представлен в данном выпуске (см. программу "Крибедж").

Кроме тома настольных игр готовятся тома стратегических, аркадных, адвентюрных и

обучающих программ.

Мы особенно рады, что сможем удовлетворить значительно выросший интерес читателей к адвентюрным программам, дело в том, что печать этих программ сопряжена с одним большим неудобством. Ведь человек, набравший текст программы на компьютере уже не сможет в нее полноценно играть, т. к. во время набора узнает все хитрости и загадки. Разработанная нами система "скрэмблинга" исходного текста позволяет уйти от этого недостатка, т. к. теперь набор текстовых сообщений идет в закодированном виде.

Наши предложения.

Теперь рассмотрим, как мы представляем себе организацию работы.

Заинтересованным организациям и предпринимателям мы передаем дискеты для IBM-совместимых машин в формате MS DOS, на которых записан готовый отформатированный текст книги. Если по ходу текста есть рисунки, то они содержатся на тех же дискетах отдельно в формате .PCX файлов.

Таким образом, мы передаем будущему издателю готовый оригинал-макет книги, распечатать который дело нескольких часов, вклеить рисунки и можно его отдавать в типографию.

Наши условия:

1. Издатель не вносит никаких изменений в исходный текст без согласования с авторами.

2. Желая дополнить текст какими-либо материалами, например рекламными, издатель прилагает их только в конце книги и от своего имени.

3. Издатель обязуется не предпринимать никаких целенаправленных усилий для издания или распространения книги за пределами того региона, который он представляет.

В свою очередь "ИНФОРКОМ" гарантирует, что лицензия на издание и распространение данной книги в этом регионе не будет предоставлена другой организации. Лицензия выдается на каждую книгу (том) отдельно по мере ее готовности.

В то же время, "ИНФОРКОМ" вставляет за собой право индивидуального обслуживания своих клиентов единичными экземплярами, в каком бы регионе они не проживали.

4. Издатель сам определяет тираж и предполагаемую розничную цену издания.

5. Финансовые расчеты с " ИНФОРКОМом" выполняются в два этапа.

Издатель оплачивает от 3% до 11% от предполагаемого объема реализации авторскому коллективу в виде авторского гонорара, после чего получает готовый текст на дискетах и лицензию от "ИНФОРКОМа" на работу в своем регионе.

После выхода тиража книги издатель бесплатно передает "ИНФОРКОМу от 100 до 400 авторских экземпляров готовой книги.

Мы ждем реакции с мест на наши предложения. Представленный разброс в процентах авторского гонорара (от 3 до 11) и в поставках готовой продукции (от 100 до 400) является объектом конкретный прямых переговоров и действует по принципу "чем больше планируемый объем реализации, тем меньше процент" и "чем больше тираж, тем больше пакет авторских экземпляров".

Пишите, телеграфируйте. Вам будет незамедлительно дан телефон для прямых контактов.

В условиях развала бывшего Союза, коллапса единой банковской системы и нарушения нормальной работы почты мы полагаем, что наше предложение вызовет должную заинтересованность. К примеру, у нас просто нет никаких других способов обслужить республики Прибалтики и ряд других регионов.

В соответствии с распределением любителей ПК "Спектрум" мы составили проспект лицензирования по регионам. Отныне в будущих номерах "ZX-РЕВЮ" мы будем регулярно освещать ход дел в этой области.

Ниже мы приводим карту регионирования. Регионы расположены в порядке убывания критерия перспективности (оценка субъективная). Москва и область не представлены, на них лицензия не выдается, здесь мы будем работать сами.

С.-Петербург и обл.
Екатеринбург и обл.
Красноярский край.
Казахстан
Тюменская обл.
Челябинск и обл.
Кемеровская обл.
Киев, Чернигов и обл.
Донецк и обл.
Минск и обл.
Брест и обл.
Мурманск и обл.
Новосибирск и обл.
Днепропетровск и обл.
Крым
Самара и обл.
Нижний Новгород и обл.
Респ. Саха (Якутия)
Хабаровский край
Пермь и обл.
Харьков и обл.
Ростов и обл.
Краснодарский кр.
Рязань, Тамбов, Воронеж, Липецк и области
Приморский край (Владивосток)
Иркутская обл.
Винница, Черновцы, Тернополь, Ивано-Франковск, Хмельницкий и области
Латвия
Респ. Коми
Орел, Курск, Белгород
Ярославль, Иваново, Кострома и области
Архангельск и обл.
Томск и обл.
Алтайский край
Магаданская обл.
Башкортостан (Уфа)
Татарстан (Казань)
Чувашия (Чебоксары)
Львов, Ужгород
Удмуртия (Ижевск)
Ставропольский край
Запорожье
Одесса и обл.
Литва
Узбекистан
Камчатская обл.
Саратов и обл.
Волгоград, Астрахань
Луганск
Омск

Чита
Тверь
Респ. Молдова
Респ. Карелия
Гродно и обл.
Могилев и обл.
Калининградская обл.
Калужская обл.
Полтавская обл.
Кировоградская обл.
Эстония
Херсон и обл.
Николаев и обл.
Ульяновская обл.
Брянская обл.
Вологодская обл.

Конечно, здесь представлена далеко не вся география. Это только крупнейшие центры, в которых наше дело развито. Мы с радостью установим связи с предпринимателями и из других регионов.

И последнее. Обращаем внимание крупнейших производителей компьютеров, что приложение хороших книг к Вашей продукции может очень сильно способствовать повышению ее конкурентоспособности.

Желаем успехов!

Ваш "ИНФОРКОМ".

BETA BASIC

Начало см. стр. 3, 47, 91

34. LIST FORMAT число.

Эта команда использует сразу два ключевых слова БЕЙСИКа и предназначена для того, чтобы управлять форматом листинга Вашей программы, т.е. с ее помощью можно получить распечатку программы в удобном для восприятия виде.

Команда имеет несколько режимов, которые задаются параметром. Начальное состояние после загрузки Бета-Бейсика - LIST FORMAT 0.

LIST FORMAT 0.

Эта команда дает распечатку, похожую на ту, которую мы получаем в стандартном БЕЙСИКе, но есть незначительное отличие, заключающееся в том, что если строка программы длиннее, чем строка экрана, то перенос ее на вторую экранную строку выполняется со сдвигом. Таким образом, левые пять столбцов экрана содержат только номера программных строк и текст становится более разборчивым и удобочитаемым.

LIST FORMAT 1.

Распечатка по этой команде делается таким образом, что каждый оператор печатается с новой строки. Более того, некоторые из операторов (см. ниже) печатаются со смещением вправо на одну позицию, что позволяет распечатывать программу "лесенкой", так как это принято в языках, поддерживающих структурное программирование, например "ПАСКАЛЬ", "СИ" и др.

LIST FORMAT 2.

Действие то же, что и для LIST FORMAT 1, но автоматическое смешение для некоторых операторов выполняется не на одну, а на две позиции вправо.

LIST FORMAT 3.

Действие то же, что и для LIST FORMAT 0, но печать программы выполняется без номеров строк.

LIST FORMAT 4.

Действие то же, что и для LIST FORMAT 1, но печать программы выполняется без номеров строк.

LIST FORMAT 5.

Действие то же, что и для LIST FORMAT 2, но без номеров строк.

Для глаза приятнее воспринимать смещение "лесенкой" на две позиции, но поскольку у нас невелика ширина экранной строки - всего 32 символа, то ее может оказаться недостаточным и поэтому введены режимы 1 и 4.

Появление в тексте программы следующих операторов вызывают автоматический сдвиг листинга:

DEF PROC, DO, FOR - сдвигают все последующие операторы вправо на один или на два символа до тех пор, пока не встретятся соответствующие им END PROC, LOOP или NEXT.

Операторы IF, ON ERROR и ON сдвигают все прочие операторы своей программной строки вправо.

Операторы ELSE и EXIT IF отменяют сдвиг текущего оператора на одну или две позиции.

Если Вам захочется, чтобы оператор после ELSE или THEN печатался с новой строки, то Вы можете после них поставить двоеточие.

Пример действия команд LIST FORMAT 0 и LIST FORMAT 2 показан ниже.

```
100 DEF PROC primer
110 FOR n=1 TO 10: PRINT "primer": NEXT n
120 DO: PRINT a$: INPUT b$
130 PRINT "abc": LOOP
140 IF x=1 THEN: PRINT "yes": PAUSE 50: ELSE: PRINT "no"
150 END PROC
```

```
100 DEF PROC primer
110   FOR n=1 TO 10
          PRINT "primer"
      NEXT n
120   DO
          PRINT a$
          INPUT b$
130       PRINT "abc"
      LOOP
140   IF x=1 THEN
          PRINT "yes"
          PAUSE 50
      ELSE
          PRINT "no"
150   END PROC
```

Если Вы очень привыкли к стандартному БЕЙСИКУ, то первое впечатление будет, что второй листинг выглядит несколько странно, но немного практики и вы почувствуете, насколько он удобнее.

35. LIST PROC имя процедуры.

Команда позволяет распечатать не всю программу, а только процедуру, которую Вы задали именем, например LIST PROC box.

На практике эта команда очень часто используется в длинных программах. Тогда Вам не надо помнить, в каком месте программы расположена Ваша процедура.

Может быть, Вам захочется хранить имя процедуры в строковой переменной. Команда LIST PROC не сможет тогда ее обработать напрямую и требуется обходной прием:

```
10 INPUT a$: KEYIN "LIST PROC"+a$
```

Две системные переменные содержат номера первой и последней строк процедуры, заданной в LIST PROC. Их адреса 23635 и 57358, соответственно. Работа с ними иногда может быть очень полезной и ее удобно выполнять, используя функцию DPEEK.

36. LIST REF.

REF - ключевое слово на клавише "SHIFT"+"7".

См. также REF.

LIST REF - дает список номеров помеченных строк. Меткой может быть имя переменной, число или набор символов. LIST REF тесно связана с командой REF и описана более подробно в соответствующем разделе, который желательно предварительно прочитать.

Если в некоей программной строке метка существует не один раз, то при печати списка этот номер строки появится многократно

Например:

```
10 FOR n=1 TO 10: PRINT n
20 NEXT n
```

Для такой программы команда LIST REF n даст следующий список:

```
10
10
20
```

Чтобы направить список на принтер, Вы можете использовать команду LLIST REF или:

```
LIST #(номер потока) REF метка
```

37. LOCAL переменная <,переменная><,переменная>...

Клавиша: "SHIFT" + "3"

См. также главу 3 "Процедуры" и раздел "DEF PROC".

Команда LOCAL позволяет создавать специальные переменные, которые существуют только внутри заданных процедур. (Параметры процедур автоматически имеют статус локальных и потому не нуждаются в объявлении их оператором LOCAL.) Использование команды LOCAL для создания локальных переменных в процедурах имеет то преимущество, что исключает образование помех, вызванных тем, что процедура может изменить содержимое какой либо переменной, относящейся к главной программе. Благодаря этой команде процедура может иметь переменные x и a$ и изменять их как угодно, не изменяя переменные x и a$ в главной программе.

Команда LOCAL может быть использована только в теле объявленной процедуры, иначе система выдаст сообщение об ошибке "Missing DEF PROC". Самое удобное для нее место - после объявления процедуры, сразу за DEF PROC. Тогда при чтении распечатки сразу становится ясно, какие переменные в процедуре используются. По желанию Вы можете иметь в процедуре несколько команд LOCAL.

Если переменные, объявленные как LOCAL, до этого объявления в программе уже существовали, то они "прячутся". На самом деле они, конечно существуют в области программных переменных и команда CLEAR их не уничтожит, но с точки зрения процедуры их как бы нет. В процедуре теперь эти имена могут использоваться снова. В конце работы процедуры все переменные, объявленные как LOCAL, будут уничтожены и исходные значения (если они были спрятаны) будут восстановлены. Вот простой пример.

```
10 LET n=1
20 test: REM чтобы выйти из курсора "K" воспользуйтесь ведущим пробелом.
30 PRINT n
40 STOP
100 DEF PROC test
110 LOCAL n
120 LET n=999
130 PRINT n
140 END PROC
```

Запустите этот пример и убедитесь, что n в процедуре равно 999, а вне ее n равно 1. Если вы опустите строку 120, то n не будет существовать внутри процедур, а если Вы опустите строку 10, то n не будет существовать в главной программе.

Если наша процедура test будет внутри себя вызывать некую субпроцедуру (например из строки 125), то для нее переменная n со значением 999 будет глобальной, для этой субпроцедуры процедура test является как бы главной программой.

Все переменные, доступные в процедуре являются глобальными по отношению к процедурам, вызываемым из нее. Чтобы "спрятать" эти переменные в нижележащих процедурах. Вы можете и в них использовать оператор LOCAL или использовать их в списке параметров.

В отличие от многих прочих диалектов БЕЙСИКа, БЕТА-БЕЙСИК поддерживает локальные массивы. Если, к примеру, в главной программе Вы используете a$, то в процедуре командой LOCAL a$ все строковые массивы a$ или переменные a$ будут локальными и не повлияют на переменные главной программы. Если Вы хотите числовой массив сделать локальным, то после его имени надо использовать круглые скобки. Например, LOCAL b(). Это "спрячет" любой существующий массив b(). После этого Вы можете в процедуре создать свой новый локальный массив b() того размера, который Вам нужен.

38. LOOP

или LOOP WHILE условие

или LOOP UNTIL условие

клавиша: L

См. также DO, EXIT IF.

LOOP - это нижняя граница цикла DO-LOOP (см. DO). Команда LOOP, заданная сама

по себе вызывает возвращение исполнения программы к соответствующему оператору DO.

LOOP может употребляться вместе с кодификаторами WHILE и UNTIL, которые обеспечивают возврат в начало цикла по условию.

LOOP WHILE условие - обеспечивает возврат назад, к вершине цикла только в том случае, если <условие> справедливо, в противном случае выполняются операторы и строки, стоящие за LOOP.

LOOP UNTIL условие имеет противоположное действие. Если указанное <условие> справедливо, возврат к вершине цикла не выполняется, а выполняется, когда оно ложно.

Если Вы используете в программе оператор LOOP без соответствующего DO, то получите сообщение об ошибке: T,"LOOP without DO".

39. MERGE

См. также DEFAULT = устройство. Эта команда имеет дополнительную возможность, которую использовать смогут только немногие владельцы микродрайва. Она позволяет загружать без автозапуска те программы, которые выполнены автостартующими. С ленты это можно делать и так.

40. MOVE

Эта команда тоже развивает возможности владельцев микродрайва. Она теперь позволяет перемещать не только блоки данных, но и программы, машинный код и массивы.

Чтобы перенести файл "test" с драйва 1 на драйв 2, делайте так: MOVE "m"; 1; "test" TO "m"; 2; "test"

Команда сработает, независимо от физической природы файла. Хоть MOVE и выглядит удобной командой, на практике большие файлы быстрее перемещаются через SAVE/LOAD.

41. ON

Клавиша: O

БЕТА-БЕЙСИК обеспечивает две различные формы оператора ON.

Вы можете задать после ON список число или выражение, которое определит, к какой строке будет выполняться переход по оператору GO TO или GO SUB.

Это довольно традиционный оператор для различных диалектов БЕЙСИКа, хотя и довольно архаичный в эпоху широкого применения процедур.

Вторая форма позволяет не только избрать строку, но и оператор.

Пример. Первая форма.

```
GO TO ON число, номер строки, номер строки, номер строки...
```

или

```
GO SUB ON число, номер строки, номер строки, номер строки...
```

(Более стандартной выглядела бы форма ON число GO TO строка..., но клавиатурная система "Спектрума" делает ее усложненной).

Обычно при программировании на "Спектруме" такой многопозиционный переключатель организуют так:

```
10 INPUT choice:
      GO TO choice*100+100
```

Но с использованием ON это можно делать более гибко, т.к. номера строк перехода не должны быть числами одного заданного ряда.

```
10 INPUT choice:
      GO TO ON choice: 90,135,60,40
20 PRINT "Enter 1 to 4": GO TO 10
```

В этом примере будет выполнен переход к строке 90, если choice равно 1, к строке 135, если choice равно 2, к строке 60, если choice равно 3 и т.д. Если choice не входит в диапазон от 1 до 4, то никакого перехода не делается, а выполняется следующей оператор или программная строка, а она предоставляет пользователю возможность повторить свой

выбор. В этом примере вы можете заменить INPUT на GET и тогда получите элегантную технику для написания программ, управляемых от меню.

Вторая форма.

```
ON число: оператор: оператор:...
```

Эта форма позволяет исполнить тот или иной оператор из списка, в зависимости от того, какое значение принимает параметр <число>.

Из списка операторов исполняется только указанный, после чего программа переходит к исполнению следующей строки. Если параметр <число> больше, чем количество операторов в строке, то сразу выполняется переход к следующей строке. Ниже приведен пример использования этой команды в ответ на ввод от пользователя числа 1,2 или 4. Если он введет число 3, то никакой реакции не произойдет и программа продолжится в естественном порядке. Это обеспечено тем, что третий оператор в строке 20 - пустой.

```
10 INPUT x
20 ON x:  PRINT "one":
          PRINT "two"::
          PRINT "four"
30 GO TO 10
```

Совсем необязательно, чтобы операторы в строке ON были одного типа. Ниже показан пример использования "смешанных" конструкций, содержащих и вызов процедуры и GO SUB и PRINT.

```
10 DO
20    GET number
      ON number
          GO SUB 100
          sound
          GO SUB 200
          PRINT "bye"
30 LOOP
```

Использование ON со списком процедур - наиболее современная, рациональная и удобочитаемая форма.

42. ON ERROR номер строки

или

ON ERROR: оператор: оператор:...

Клавиша: N.

Для оператора ON ERROR возможны две формы записи. Первая форма задает номер строки, к которой происходит переход, если происходит прерывание работы по ошибке. Остальные операторы в строке ON ERROR в этом случае не имеют к нему специального отношения.

Во второй форме после ошибки начинается выполнение операторов, содержащихся в данной строке.

За ошибку считается любая ошибка из числа перечисленных в инструкции к "Спектруму" или из числа приведенных в приложении к инструкции по работе с языком "БЕТА-БЕЙСИК 3.0", т.е. все сообщения Бейсика, кроме сообщений:

```
0: "OK"
9: "STOP statement"
```

Этот режим можно отключить оператором ON ERROR 0, но он также отключается автоматически при работе процедуры обработки ошибки и вновь включается после возврата в главную программу. (Ей приходится отключаться, иначе был бы конфуз, когда при обработке собственной ошибки она вызывала бы саму себя и вновь сталкивалась с той же ошибкой.)

Этой процедуре доступны три специальные переменные, которые могут быть очень и очень полезны: LINO, STAT и ERROR. Все это не ключевые слова и набираются по буквам.

LINO и STAT - номер строки, в которой произошла ошибка и номер оператора в строке.

ERROR - номер кода ошибки, когда мы закончим печать инструкции мы дадим коды всех ошибок.

Вы можете пользоваться этими переменными в своих целях, но делать это надо до того, как будет активизирована процедура ON ERROR, потому как при ее активизации, как и при активизации режима TRACE (см. ниже), эти переменные могут быть перезатерты.

Не рекомендуем Вам все возможные ошибки обрабатывать через ON ERROR. Желательно все, за исключением одного-двух сообщений обрабатывать естественным путем, иначе при отладке программы может быть трудно разобраться с тем, что в ней происходит. Во всяком случае Вы ведь не предполагаете, что Ваши программы будут изобиловать многочисленными ошибками. В нижеприведенном примере программа печатает точки на экране, а процедура обработки ошибок отсекает те, которые выходят за его пределы.

Первая форма выглядит так

```
100 ON ERROR 5000
110 FOR n=1 TO 10:
      INPUT "x coord "; x;
      "y coord "; y
120 PLOT x,y: NEXT n
.....................
4990 STOP
5000 IF error=11 AND lino=120
      THEN RETURN: ELSE POP:
      CONTINUE
```

Обратите внимание: Оператор STOP стоит в программе для того, чтобы предотвратить случайное исполнение процедуры обработки ошибки. Значения LINO и ERROR проверяются потому, что сообщение "Integer out of range" является широко распространенным и появляется во многих случаях. Возврат из процедуры обработки ошибки выполняется к оператору, следующему за тем, который вызвал ошибку, поэтому возврат выполняется к оператору NEXT n. Если номер строки или код ошибки были не те, то выполняется оператор CONTINUE. В результате этого подозрительный оператор выполняется еще раз (кроме случая "BREAK into program") и, поскольку CONTINUE не задействовал ON ERROR, то теперь обработка ошибки пойдет стандартным путем. Оператор POP очищает стек от адреса возврата в главную программу. Если этого не сделать, то нормальная работа стека будет нарушена.

Вторая форма выглядит так:

```
100 ON ERROR
      IF error=11 AND lino=120
          THEN RETURN
      ELSE
          POP
          CONTINUE
```

Есть одна "ошибка", которую необходимо обрабатывать путем, отличным от прочих. Это "BREAK into program". Поскольку ON ERROR отключается, когда выполняется переход GO SUB к процедуре обработки ошибки, то обычное действие BREAK будет состоять в том, что Ваша процедура остановится после исполнения первого же оператора, поскольку Вы еще не успели отпустить клавишу BREAK. Поэтому, если Вы хотите, чтобы по нажатию BREAK были предприняты какие-то действия и ваша процедура обработки ошибок работала бы, то вам необходимо ввести паузу в первом операторе процедуры обработки ошибки, достаточную для того, чтобы пользователь освободил клавишу BREAK. Для этой цели неплохо может служить оператор BEEP. Если вы не хотите, чтобы генерируемый звук был слышен, используйте те частоты, которые не слышны для человеческого уха. Например:

```
100 ON ERROR 5000
110 PRINT "round and ";:
      PAUSE 10: GO TO 110
.............................
4990 STOP
5000 IF error = 21
      THEN BEEP 1.69:
```

```
BORDER RND*7:
RETURN: ELSE POP:
CONTINUE
```

43. OVER 2.

См. также GET, PLOT.

Кроме 0 и 1 OVER теперь может работать с параметром 2. OVER 2 имеет то действие, что те символы или рисунки, которые печатаются командой PLOT <строковая переменная>, добавляются к тому, что уже есть на экране, а не затирает имеющееся изображение и не инвертируют общие точки. Пиксел приобретает цвет INK, если он был INK или он становится INK от нового рисунка. Таким образом, OVER 2 аналогичен слиянию изображений по логике OR, в то время как OVER 1 работает по логике XOR. (Различие в их логике будет описано в разделе "функции").

44. PLOT X,Y <;строка>

См. также: GET, OVER 2, CSIZE, управляющие коды.

Как видите, Бета-БЕЙСИК допускает выполнение PLOT не только для точек, но и для символьных строк. Это могут быть обычные символьные строки или экранные блоки, снятые с экрана с помощью GET и сохраненные в памяти, как строковые переменные.

Координаты, выступающие параметрами оператора PLOT имеют отношение к левому верхнему углу той символьной последовательности, которая помещается на экран. Могут использоваться и все обычные прочие квалификаторы оператора PLOT, такие, как INVERSE, OVER, INK и т.п. Если печатаемая строковая последовательность выходит за пределы правого нижнего угла экрана, то она появляется в левом верхнем углу. Если какая-то часть помещаемого на экран символа выходит за его пределы, выдается сообщение об ошибке "Integer out of range". В то же время, нижние 7 пикселов изображаемых символов могут выходить за нижнюю границу и в этом случае они изображаются в системном окне экрана.

Координаты позиции PLOT, которые используются в качестве стартовых для работы оператора DRAW, не нарушаются при использовании команды PLOT со строковой переменной.

Изменяя координаты позиции PLOT для символа, Вы можете добиться гораздо более плавного эффекта мультипликации, чем это возможно в обычном БЕЙСИКе при использовании оператора PRINT AT.

```
100 FOR x=16 TO 224: PLOT x,x/2; "<>": NEXT x
```

Попробуйте ввести в цикл параметр STEP 2 или 3 или более. Скорость будет выше, а эффект не совсем тот. Поскольку рисунок "<>" имеет вокруг себя поле цвета PAPER шириной по крайней мере в один пиксел, то при движении по экрану он будет сам себя стирать, если перемещение делать на один пиксел за шаг. Некоторые буквы, например "T" имеют включенные пикселы цвета INK, подходящие к самой кромке знакоместа, поэтому существуют такие направления движения символа, при котором он не стирается, а оставляет след на экране. Если Вы сами конструируете себе символьный набор, то не помешает делать его так, чтобы со всех сторон символа оставалось поле шириной в один пиксел.

Вы можете увеличивать или уменьшать размеры помещенных на экран символов, графики, блоков и т.п. с использованием команды CSIZE (см. соответствующий раздел). Команда CSIZE должна непосредственно следовать за PLOT. Например:

```
PLOT CSIZE 32;INK 2: 100,88;"HI!"
```

В печатаемую символьную строку Вы можете включить управляющие коды CHR$ 8 - CHR$ 11. Тогда Вы сможете получать гораздо более сложные геометрические построения.

Возможность помещать строки на экран командой PLOT очень полезна для печати графиков и диаграмм. Во-первых, здесь возможна более высокая точность, а во-вторых можно пользоваться той же координатной системой, которой пользуются графические функции, например DRAW, CIRCLE.

45. POKE адрес, строка

БЕТА-БЕЙСИК дает возможность не только выполнять POKE для чисел, но и для символьных строк, что в совокупности с функцией MEMORY$ дает возможность скоростных манипуляций с большими массивами памяти. Надо, правда, отметить, что если нерасчетливый POKE какого-либо числа может вывести из строя программу, то для строк это становится еще более критичным.

Рассмотрим пример:

```
10 LET screen=16364
20 POKE screen, STRING$ (6114, "U")
```

О функции STRING$ мы будем говорить позже, в разделе "Функции". Здесь мы заполняем экранную память кодом буквы "U", но воспринимается это не как код, а как двоичное число 01010101, что изображает на экране полосы.

Следующий пример демонстрирует копирование начальной области ПЗУ в файл экранных атрибутов.

```
10 LET attr = 22528
20 POKE attr, MEMORY$()(1 TO 704)
```

Теперь давайте нарисуем на экране что-нибудь простое, сохраним изображение в строковой переменной и затем восстановим его на экране, используя POKE.

```
10 CIRCLE 128,88,70
20 FILL 128,88
30 LET a$ = MEMORY$() (16384 TO 23295): REM весь экран
40 CLS: PRINT "нажмите любую клавишу": PAUSE 0
50 POKE 16384,a$
```

В памяти компьютера могут одновременно храниться несколько таких картинок, вы можете оперативно менять их местами и по одной выбрасывать на экран. Если Вам надо больше картинок, вы можете одну треть экрана считывать в строковую переменную. Если вас интересует не только черно-белая информация, но и атрибуты цвета, то и треть файла атрибутов Вы тоже можете сбрасывать в такую же переменную.

Для хранения черно-белой информации экранных сегментов вы может пользоваться следующими командами:

Верх:

```
LET a$=MEMORY$()(16384 TO 18431)
```

Середина:

```
LET a$=MEMORY$()(18432 TO 20479)
```

Низ:

```
LET a$=MEMORY$()(20480 TO 22527)
```

У компьютера достаточно памяти даже для того, чтобы исполнить реальный мультфильм путем использования команды POKE в сегмент экрана. Чтобы хранить данные в памяти, можно использовать массив DIM a$(10,2048).

Конечно, потенциальных возможностей у манипуляций с большими объемами памяти гораздо больше, чем просто обслуживание экранной памяти. Вы можете, например очистить большой объем верхней памяти и сохранить там целую программу, а потом вызывать ее опять. Так можно обеспечить одновременное присутствие в памяти компьютера сразу нескольких программ.

```
CLEAR 33900
10 POKE 34000, MEMORY$() (22552 TO 33800)
20 REM остальные строки этой программы.
```

Теперь Вы можете сделать NEW и удалить Вашу программу, но ее копия останется в верхней памяти выше границы RAMTOP, установленной оператором CLEAR.

Можно опять вызвать эту программу в работу:

```
POKE 33552, MEMORY$() (34000 TO 44248)
```

Эту вызывающую команду можно "подвесить" на клавишу, определяемую пользователем и, поскольку такие определения тоже хранятся выше уровня RAMTOP, то оно будет защищено от разрушения командой CLEAR.

```
DEF KEY "j": POKE 23552, MEMORY$() (34000 TO 44248)
```

В тот момент, как программа будет восстановлена, она продолжит работу с того места, в котором она была "спрятана", т.к. вместе с программой отгружались и восстанавливались все ее системные переменные.

Итак, как видите, БЕТА-БЕЙСИК дает Вам несложный механизм организации виртуального диска (RAM-диска) для работы сразу с несколькими программами.

И еще одна идея для использования POKE, заключающаяся в возможности сохранения машинного кода вместе с БЕЙСИК-программой. Присвойте этому блоку кодов имя символьной переменной и выгрузите БЕЙСИК на ленту таким образом, чтобы при последующей загрузке POKE, находящийся в строке автостарта, автоматически перебрасывал этот код на нужное место памяти и дальше запускайте его. Это делается значительно быстрее, чем раздельная загрузка БЕЙСИК-программы и блока машинных кодов. Для того, чтобы освободить место под блок машинного кода и не потерять при этом БЕЙСИК-переменные, воспользуйтесь командой CLEAR.

46. POP <числовая переменная>

Клавиша: Q

Команда POP удаляет последний адрес со стека, обеспечивавшего правильную работу команд GO SUB, DO-LOOP, PROC. Если при этом Вы используете параметр <числовая переменная>, то номер строки, к которой должен был бы быть исполнен переход, если бы вы не сняли его со стека, запоминается в этой переменной.

Команда POP может позволить Вам завершать работу в подпрограммах, процедурах и циклах не естественным путем и при этом закупорки стека не произойдет, если выходя в неположенном месте, например из цикла, Вы снимете адрес естественного возврата со стека. Команда POP без параметра просто удалит этот адрес со стека, а если Вы используете ее с параметром, например с переменной loc, то этот адрес еще останется в вашей переменной и, может быть, он Вам впоследствии для чего-нибудь пригодится. Может быть, по ходу программы Вы примете решение о том, что POP был сделан неправильно и надо все-таки перейти туда, куда должна была возвращать Ваша процедура. В этом случае знание loc позволит Вам сделать GO TO loc+1, хотя надо отметить, что это все же не вполне то же самое, что и естественный RETURN. Дело в том, что адрес, откуда Вы прошли в подпрограмму, запомнен на стеке и после RETURN Вы возвращаетесь туда, откуда уходили. Возвращаетесь либо на следующую строку, либо на следующий оператор в той же строке, если он есть. Переход же по GO TO loc+1 не может вернуть Вас к следующему оператору, а только к следующей строке.

Пример:

```
100 GO SUB 500
110 STOP
500 POP loc
510 PRINT "Подпрограмма вызывалась из строки "; loc
520 GO TO loc
```

Если Вы в строке 520 поставите

```
520 RETURN
```

то получите сообщение "RETURN without GOSUB", поскольку к этому моменту на стеке уже не будет никакого адреса, ведь он был снят командой POP.

Вызов команды POP, когда на стеке нет данных, дает сообщение об ошибке V, "No POP data".

47. PROC имя <параметр><,параметр><,параметр>...

Клавиша: 2

См. также Главу 3 "Процедуры", а также раздел DEF PROC.

Для того, чтобы вызвать процедуру, в версии 3.0 нет необходимости давать ключевое слово PROC и оно сохранено главным образом для того, чтобы обеспечить совместимость с предыдущими версиями БЕТА-БЕЙСИКа. Вы можете обращаться к процедуре по ее имени без этого ключевого слова, для чего имя вводятся после отключения K-режима вводом

ведущих пробелов или вызовом режима KEYWORDS 4.

С другой стороны, слово PROC используется, например, в команде LIST PROC (см.).

48. READ LINE строковая переменная <,строковая переменная>...

См. также Главу 3 "Процедуры".

Команда состоит из двух ключевых слов и позволяет работать без кавычек со списком данных, которые обычно должны иметь кавычки. Например:

```
100 DATA dog, rat, fish, frog, z$,12,"?*+"
110 READ LINE a$ 120 PRINT a$: GO TO 110
```

Оператор DATA ограничивает типы данных, которые можно хранить таким способом, поскольку допустимы только полноценные выражения. В строке 100 "cat" может быть числовой переменной, но "?*+" требует наличия кавычек. Оба типа можно было бы объединить, добавив:

```
115 IF a$(1)=CHR$ 34 THEN LET a$=VAL$ a$
```

READ LIKE позволяет сделать строки DATA более удобными для чтения, хотя главное назначение этой команды - сделать возможным ввод строковых данных без кавычек.

49. REF метка

Клавиша: SHIFT + "7"

Эта команда позволяет выполнить в программе поиск заданной "метки", которая может быть числом, именем переменной или набором символов.

Когда указанная метка найдена, строка, содержащая ее, появляется в системном окне компьютера (в строках редактирования). При этом курсор стоит непосредственно за меткой. Если Вы не хотите вносить изменений в найденную строку. Просто нажмите ENTER. Чтобы найти другие строки, содержащие такую же метку, нажмите ENTER еще раз и так далее, когда поиск будет закончен полностью, появится сообщение "O.K. "

Если вместо нажатия ENTER вы введете какую-либо команду, то компьютер решит, что Вы закончили поиск и теперь, чтобы опять продолжить поиск. Вам надо снова задавать команду REF.

Примеры: (символы, не являющиеся буквой, цифрой и знаком $ показаны, как _.)

```
REF a$     ищется a$
REF count        ищется _count_
REF "count"      ищется count
REF 1            ищется 1 (число)
REF "1"          ищется 1 (символ)
REF 12*4         ищется 12 (число) * 4 (число)
REF (a$)         ищется значение a$, так, например, если a$ = "fish", то ищется "fish", а не
       a$.
REF x            ищется значение x, например, если x=10, то ищется 10(числовая форма).
```

Не играет никакой роли регистр встреченных символов - прописные буквы или строчные.

При поиске переменных необходимое условие, что имя должно начинаться и заканчиваться символами, отличными от буквы или цифры, Это позволяет отличать разные переменные, имеющие общие символы в имени, например:

```
count account counts
```

При поиске чисел поиск идет не по их символьному представлению на экране, а по их числовой интегральной (пятибайтной) форме, которая стоит за всяким числом, появляющимся в любой БЕЙСИК строке. Это тоже позволяет избежать конфузов. Таким образом, внутренние вложения в переменные и числа не мешают правильной работе.

Если же Вы ищете набор символов, заключите его в скобки и он будет найден, даже если является вложением в большую строку.

Если Вы используете строковую переменную при поиске, то ее надо заключить в скобки, чтобы БЕТА-БЕЙСИК мог отличить случай поиска по имени от поиска по значению.

50. REF имя переменной.

Клавиша: SHIFT + "/"

Мы рекомендуем Вам предварительно прочитать главу 3 "Процедуры", прежде чем двигаться дальше.

Здесь ключевое слово REF используется для того, чтобы сообщить процедуре о том, что такие-то и такие-то параметры этой процедуры передаются в виде ссылки, а не по значению.

Это означает, что во время исполнения процедуры соответствующие переменные могут быть временно переименованы в имя, стоящее после REF. Это обеспечивает передачу параметров не только из вышележащей процедуры в нижележащую, но и наоборот снизу вверх. Пассивы всегда должны передаваться, как ссылки. В нижележащем примере REF a$ относится к символьной строке или массиву, а REF b() - относится к числовому массиву.

```
100 DEF PROC crunch REF a$, REF b(), REF OUTPUT
```

51. RENUM <*><начало TO конец> LINE <новое начало> STEP <шаг>.

Клавиша: 7

RENUM обеспечивает очень мощные возможности по перенумерованию блоков программных строк, их перемещению и их копированию.

Просто команда RENUM без параметров перенумерует все строки программы так, что номером первой строки будет 10, а далее все строки будут идти с шагом по 10. Но можно и задать перенумерацию не всей программы, а только заданного блока строк.

RENUM 130 TO 220 - перенумерует все строки из этого диапазона.

RENUM 130 TO - перенумерует все строки, начиная со строки 130 и до конца программы.

RENUM TO 100 - перенумерует строки, начиная с первой программной строки и до строки 120 включительно, за исключением нулевой строки.

Перенумерованный блок будет перемещен в программе на заказанное место, если в памяти для него место имеется, в противном случае может быть выдано сообщение об ошибке: G, "No room for line" ("Для строки нет места").

Команда RENUM* отличается тем, что она выполняет не перемещение строк, а их копирование, т.е. создается новый блок с заданной нумерацией строк, но старый при этом не уничтожается.

Вы можете задать в качестве первой строки не десятую, а любую другую, указав параметр LINE (это ключевое слово). Если Вас не устраивает стандартный шаг нумерации строк в БЕЙСИКе через десять, задайте свой в качестве параметра STEP (ключевое слово). И LINE и STEP могут быть даны или опущены по желанию, но если Вы их даете, то порядок их следования должен быть только таким, как указано в описании команды и не наоборот.

Некоторые примеры:

```
RENUM
RENUM LINE 100 STEP 20
RENUM 100 LINE 300 (перенумеровывается только одна строка)
RENUM 1540 TO LINE 2000
RENUM 100 TO 176 LINE 230 STEP 5
RENUM * 10 TO 100 LINE 500 -копирование блока строк.
```

При перенумерации программных строк все ссылки на строки по их номеру тоже перерабатываются, включая GO TO, GO SUB, RESTORE, RUN, ON, ON ERROR, TRACE, LIST, LLIST, LIKE.

Операторы DELETE и CLOCK - не перерабатываются, Вам придется заняться этим вручную.

RENUM не может также самостоятельно переработать вычисляемые ссылки на номера строк, такие как GO TO ln*100.

После завершения перенумерации все указания на такие подозрительные места будут распечатаны на экране, чтобы Вы могли с ними разобраться, например:

```
Failed at 100:2
Failed at 230:4
```

Все GO TO, GO SUB и т.п., содержащие после себя такие вычисляемые адреса перехода, будут распечатаны, даже если Вы и перенумеровываете лишь малый блок и в ней нет таких операторов, потому что они могут быть в других местах программы и указывать именно на тот блок, который Вы и перенумеровали.

Если Вы желаете направить распечатку таких сложных строк на принтер, воспользуйтесь:

```
OPEN #2, "P": RENUM:
OPEN #2, "S"
```

Примечание: во время работы команда RENUM образует временный буфер в экранной области компьютера, что сопровождается неожиданными помехами на экране. Не обращайте на них внимание, по окончании работы экран будет восстановлен.

52. ROLL код направления <, число><; x, y; ширина, длина >

Клавиша: R

См. также SCROLL

Команда ROLL перемещает изображение, имеющееся на экране или в заданном окне вверх, вниз, влево или вправо. Все, что выходит за пределы окна, тут же появляется с противоположной стороны. Одним словом, команда, в отличие от команды SCROLL, не разрушает содержимое экрана, а только перемещает его.

Как видите, команда ROLL имеет довольно сложный синтаксис, хотя большая часть сопровождающих параметров служит только для того, чтобы задать окно, внутри которого действует ROLL. Если Вы, например, хотите сдвинуть на один пиксел содержимое текущего окна, а таковым обычно является весь экран, если вы не задали иначе, то команда имеет такой простой вид:

```
10 ROLL код направления
```

Если вам надо сдвинуть только черно-белую графику без цветовых атрибутов, то в качестве кода направления нужно задать 5,6,7 или 8 (соответственно это означает влево, вниз, вверх, вправо).

Поскольку команда ROLL вызывает смещение изображения на небольшую величину, самый лучший способ ее применения - внутри циклов.

Нарисуйте (DRAW, CIRCLE) какое-либо несложное, но крупное изображение или просто дайте команду LIST и экран будет заполнен, а затем попробуйте следующие строки:

```
100 FOR d=5 TO 6: FOR p= 1 TO 100
110 ROLL d
120 NEXT p: NEXT d: STOP
```

Можно делать смещение и по диагонали. В этом случае в цикле выполняется последовательность движений вверх, вправо или т. п. Если хотите, чтобы движение шло более быстро (но менее плавно), попробуйте:

```
110 ROLL d, 4
```

Этот параметр указывает на сколько пикселов за один раз должно производиться смещение. Он не должен быть более, чем 256 при горизонтальном движении и чем 177 - при вертикальном. Если параметр не указав, по умолчанию принимается единица. Очевидно, что чем больше это число, тем более значительно будет передвинуто изображение. При вертикальной движении скорость обычно пропорциональна количеству пикселов, передвигаемых за раз. При горизонтальной движении наилучший результат дает шаг в 4 или в 8 пикселов, т. к. в этом случае используются более скоростные команды машинного кода процессора Z-80.

Если Вы хотите передвигать не только чёрно-белую информацию, но и цветовые атрибуты, добавьте к коду направления число 4, а если хотите передвигать только атрибуты, наоборот отнимите 4.

Код	Направление	Действие
1	Влево	Атрибуты
2	Вниз	Атрибуты
3	Вверх	Атрибуты
4	Вправо	Атрибуты

5	Влево	Графика
6	Вниз	Графика
7	Вверх	Графика
8	Вправо	Графика
9	Влево	Граф. + Атр.
10	Вниз	Граф. + Атр.
11	Вверх	Граф. + Атр.
12	Вправо	Граф. + Атр.

Атрибуты можно сдвигать только на 8 пикселов за один шаг и параметр, определяющий шаг смешения здесь игнорируется.

При использовании кодов от 9 до 12 можно добиваться наилучшего соответствия между шагом перемещения черно-белой графики и цветовых атрибутов. Посмотрите, что происходит:

```
10 PRINT AT 10, 10: PAPER 2; "DEMO"
20 ROLL 9: PAUSE 10: GO TO 20
```

Вы увидите, что рассогласование движения атрибутов и графики достигает 4-х пикселов. Теперь попробуйте такой вариант:

```
10 PRINT AT 10, 10; INK 2; " DEMO "
```

Дополнительные пробелы, охватывающие слово "DEMO", обеспечивают то, что оно всегда прикрыто 6-символьной полосой красного цвета. Аналогичный метод работает и для других геометрических форм. Нижеприведенный пример создает цветные круги, каждый из которых окружен защитным кольцом с соответствующей установкой атрибутов.

```
10 LET y=88, r=15
20 FOR n=1 TO 4
30 LET x=n*48+8
40 CIRCLE x,y,r
50 FILL INK n; x, y
60 CIRCLE INK n; INVERSE 1: OVER 1; x, y, r+5
70 NEXT n
80 ROLL 9
90 GO TO 80
```

Если перемещению подлежит не весь экран, а только его часть (окно), то его Вы можете задать, указав координаты x и y левого верхнего угла, ширину и высоту. При этом здесь применяются разные координатные системы. Параметры X, Y и высота окна задаются в пикселах, а ширина окна - в знакоместах. Ее конечно тоже можно было бы задавать в пикселах, но тогда команда получается медленноработающей. Итак, параметр ширины может быть от 1 до 32, а параметр высоты - от 1 до 176.

Что же касается атрибутов, то они перемещаются только с точностью до знакомест, поэтому работая в цвете необходимо четко планировать свои действия.

Вы можете предварительно задать окно командой WINDOW и тогда команда ROLL будет работать с данным окном без необходимости указывать его параметры.

ROLL может эффективно применяться при создании аркадных игр для обеспечения плавных движений героя или ландшафта. Интересный головокружительный эффект может иметь создание сразу нескольких пересекающихся окон, двигающихся в разных направлениях.

```
100 LIST: LIST: LIST
110 LET pixels=4
120 ROLL 5, pixels; 0,175;32,88
130 ROLL 6, pixels; 0,175;16,176
140 ROLL 8, pixels; 0,87;32,88
150 ROLL 7, pixels; 128,175;16,176
160 GO TO 120
```

Поэкспериментируйте с этой программой. Попробуйте задать pixels=1. Попробуйте изменить строку 100 на следующую:

```
100 KEYWORDS 0: PRINT STRING$(704 , " END PROC "): KEYWORDS 1
```

Слова END PROC набирайте не по буквам, а клавишей 3 в графическом режиме. И, наконец, последний пример:

```
200 FOR n=1 TO 7: LIST: NEXT n
```

```
210 FOR m=1 TO 175:
      ROLL 5;0,175;32,m:
      NEXT m
```

(Продолжение в следующем выпуске)

Михайленко В.С.

ЗАЩИТА ПРОГРАММ

Продолжение.
(Начало см. стр. 9-16, 53-60, 97-104).

Итак, Вы осуществили подмену и загрузили Бейсик-файл под видом кодов. Теперь Вашей задачей является просмотреть этот файл и изучить его структуру. Для этого можно использовать приведенную ниже программу Бейсика:

```
1 FOR i=30000 TO (30000+"реальная длина")
2 PRINT i:TAB 7;PEEK i;TAB 11;CHR PEEK i
3 NEXT i
```

В первой строке величина "реальная длина" является своим значением для каждого конкретного случая рассматриваемой вами программы. Она определяет ту область программы, которую Вы желаете просмотреть. (Поскольку мы загрузили Бейсик-файл под видом кодов в область памяти компьютера, начиная с 30000, то это значение является исходной точкой для начала просмотра).

После того, как вы набрали текст программы дампинга и запустили ее командой RUN, то на экране появятся столбцы значений в следующем формате:

Значение ячейки памяти	Содержимое данной ячейки памяти в DEC виде	Символьное представление содержимого ячейки памяти

Чтобы Вам хорошо разбираться в сути текста, появлявшегося на экране, необходимо вспомнить структуру Бейсик-строки. Как известно, ее схематично можно представить в виде:

MM NN текст строки код ENTER (13)

где:

MM - два байта номера строки
NN - два байта длины строки

"Текст строки" почти соответствует исходному, за исключением представления цифровых величин (очень подробно информация о представлениях чисел в ZX SPECTRUM описана в трехтомнике "ИНФОРКОМа" "Первые шаги по программированию в машинных кодах") и завершает строку код ENTER 0BH (13)[1].

Когда перед Вами появится текст, будьте очень внимательны - со временем Вам будет достаточно легко читать его. В первую очередь следите за появлением кода ENTER 0BH (о его появлении свидетельствует перевод позиции печати к началу следующей строки, а также появление в строке дампинга "цифровое содержимое ячейки памяти" значение 13). После этого кода мысленно пропускайте 4 цифры (от них все равно очень мало толку, поскольку значение номера строки Бейсика из них не очевидно - это же касается и длины. Чтобы получить реальное значение, необходимо старший байт умножить на 256 и прибавить к этому значению младший байт) и внимательнейшим образом изучаете структуру Бейсика.

Для того, чтобы сделать текст Бейсик-строк исходного файла более читаемым, можно заменить строку 2 программы дампинга на следующую строку:

```
2 PRINT CHR$ PEEK i;
```

После этого Вы будете иметь картину почти аналогичную листингу, за исключением необычного представления номера строки и числовых значений.

Внимание: работа программы дампинга может быть прервана выдачей какого-либо

[1] Код ENTER равен 0Dh (Прим. OCR)

сообщения, например о неправильном цвете:

```
INVALID COLOR
```

или большом целом числе:

```
NUMBER TOO BIG
```

В этом случае необходимо набрать с клавиатуры NEXT i и дампинг продолжится.

После того, как Вы внимательно изучили структуру программы, необходимо точно определить местонахождение подпрограммы в кодах, чтобы не путать ее с последовательностью символов Бейсика. Обычно эта подпрограмма размещается после оператора REM и состоит из самых разнообразных символов.

Когда все это осуществлено, необходимо изменить содержимое этой Бейсик-программы таким образом, чтобы после ее загрузки в компьютер и запуска по команде автостарта она сама останавливалась, например по команде STOP. Это необходимо сделать таким образом, чтобы STOP сработал до блокированных POKES, которые могут не остановить работу программы, а дестабилизировать ее работу, например вызвав зависание или самосброс.

Наиболее разумным с нашей точки зрения является введение оператора STOP вместо первого оператора Бейсика. Это можно осуществить подав команду с клавиатуры

```
POKE 30004, CODE "STOP"
```

Теперь, после загрузки нашего Бейсика и запуска его на выполнение, осуществится останов по команде STOP. Наша задача выполнена. Теперь необходимо записать информацию на кассету. Пододим команду с клавиатуры:

```
SAVE "имя" CODE 30000, реальная длина
```

Однако не торопитесь включать магнитофон и нажимать ENTER. Образующийся после подачи этих команд блок будет содержать хэдер кодов и непосредственно файл кодов, аналогичный файлу Бейсика. Однако, если Вы внимательно разберетесь, то поймете, что хэдер кодов это не что иное, как "Специальный хэдер кодов (2)" т.е. фактически вам записывать его вовсе не обязательно. В данном случае на ленту можно записать только измененный файл кодов, который будет аналогичен файлу Бейсика за исключением внесенных изменений. Обозначим этот измененный файл кодов (2').

Теперь Вам необходимо загрузить измененную программу Бейсика в память компьютера. Подаем с клавиатуры команды: LOAD "" и сначала загружаем исходный хэдер Бейсика (1), после которого загружаем измененный файл кодов (2'). После загрузки программа должна остановиться с сообщением команды STOP.

Примечание: может произойти так, что измененная Бейсик-программа не остановится, а продолжит свою работу, один из возможных вариантов - это неточное внесение изменений. Дело в том, что Бейсик-файл может иметь приблизительно такую структуру:

```
1 REM - подпрограмма в кодах
2 RANDOMIZE USR - или другие команды запуска этой подпрограммы в кодах.
```

Когда Вы вносили изменения, то могли заменить код оператора REM кодом оператора STOP.

Если бы программа автостартовала со строки 1, то она, естественно остановилась бы с выдачей сообщения о выполнении оператора STOP. Однако бывают случаи, когда автозапуск осуществляется со строки 2 и, таким образом, получается, что введенный нами оператор STOP программой не обрабатывается - следовательно остановка не происходит.

Для того, чтобы исправить этот дефект, необходимо снова загрузить исходный Бейсик-файл (1') под видом файла кодов, т.е. вместе с хэдером (2). Однако теперь необходимо вносить изменения более точно, а именно в строку, которая точно обрабатывается интерпретатором Бейсика таким образом, чтобы произошел останов по выполнении оператора "STOP".

Но вот мы добились своей цели - исходная программа загружена в память компьютера без автостарта и мы приблизительно знаем ее структуру, теперь необходимо определить типы защиты, которые применены в данной программе. Здесь возможны, естественно, множество вариантов, но, тем не менее, можно с уверенностью сказать, что наиболее часто встречаются защиты, основанные на применении метода зануления

номеров строк программы, а также связанные с использованием управляющих кодов ZX SPECTRUM для сокрытия листинга исходной программы. Более подробно методы взлома в подобных случаях будут описаны в главе 3.

Теперь рассмотрим достоинства и недостатки данного метода взлома.

Одним из больших достоинств данного метода блокировки автозапуска является его доступность и простота в освоении. В самом деле, здесь Вы не используете никаких непонятных Вам программ и наиболее сложным является понять весь технологический процесс, однако, если Вам это удалось, то можно сказать, что сняты все психологические барьеры.

В то же время, несмотря на кажущуюся простоту, данный метод обладает массой недостатков. В первую очередь, это необходимость серьезной работы с магнитофоном - необходимо очень четко выставлять магнитофон перед запуском магнитной ленты, чтобы у Вас точно грузился тот или иной блок.

Кроме этого, здесь необходимо использовать дополнительное пространство на магнитной ленте для записи переделанных файлов, что не всегда удобно для пользователя.

И все же начинающим хэкерам я рекомендовал бы начинать именно используя этот метод взлома. Несмотря на некоторое неудобство, в некоторых случаях он бывает просто незаменим. Я сам достаточно долгое время работал используя исключительно этот метод и считаю, что благодаря его использованию очень многому научился. Несмотря на то, что методы, которые будут предложены Вашему вниманию в последующих разделах более совершенны и продуманны, они уступают этому методу в главном - там практически все за Вас осуществляет специальная программа и Вы фактически не участвуете во взломе. Здесь же Вы все осуществляете сами вручную и именно этот факт открывает большие перспективы.

2.2 Изменения в хэдере с использованием копировщика COPY-COPY.

Поработав достаточно длительное время, используя технологию, описанную в первой разделе этой главы, я решил усовершенствовать процесс. Основными целями при этом я поставил себе упрощение работы с кассетой и ускорение самого процесса, следует отметить, что достаточно проблематичным является создание новой методики, когда длительное время работал по другой. Но это однообразие в конце концов и помогло.

В этой работе нам понадобится копировщик COPY-COPY. Это достаточно совершенный копировщик, поскольку он имеет ряд возможностей, делающих его незаменимым в данной конкретном случае. Но, кроме всего этого, он является еще и универсальным средством взлома. Именно такое сочетание функций плюс небольшой объем памяти, занимаемый программой, и привели к необычайно широкому ее распространению. (Именно ввиду необычайной его популярности я и привожу ниже описание метода взлома, основанного на его использовании. При этом я надеюсь, что большинство читателей уже имеет в своем архиве этот копировщик, а даже если и не имеют, то ознакомившись с приведенным ниже описанием, приобретут его).

Все команды копировщика являются ключевыми словами компьютера и поэтому не набираются по буквам. Команды требуют завершения нажатием клавиши ENTER.

Приведенная ниже сводка команд разбита на блоки применения. Каждому ключевому СЛОВУ соответствует лишь одна команда копировщика. Но, в зависимости от текста, набранного после ключевого слова, выполняемые операции изменяются. Общим для каждого блока является наличие во всех его командах ключевого слова ZX SPECTRUM.

1. "CAT" - клавиша "C" - просмотр списка имен файлов на экране.

2. "LOAD" - клавиша "J" загрузка файлов в память.

LOAD - загружает программу с очередным номером (имеется в виду нумерация файлов в копировщике)

LOAD N - загружает файл на место N . Если N = 1, то загруженные перед этим файлы теряются и загрузка производится в начало рабочей области (с адреса 23296)

LOAD N TO NN - загружает файлы с номерами от N до NN.

LOAD TO N - загружает файлы от очередного номера до номера N.

LOAD AT NN - загружает файлы с адреса NN, по умолчанию файл с номером 1 загружается с адреса 23296. Можно задать NN = 23040, в этом случае величина рабочей области для загрузки файлов увеличивается до 42496 байтов. По умолчанию величина этой области равна 42240 байтов, очевидно, что NN<23040 задавать нельзя, за исключением случаев загрузки в экранную область (16384).

LOAD (NN - считывает первые NN байтов файла. Это очень удобная функция для получения стандартной копии экранов используя файлы, где загрузка экрана неразрывно сливается с загрузкой программы. Эта функция может очень помочь тем, кто имеет принтер и желает распечатывать красивые картинки. В большинстве программ картинки защищены именно таким способом (RAMBO 2,RAMBO 3, MIG 29 и т. д.) Так, например, команда

```
LOAD (6912)
```

осуществит загрузку данных в формате экрана.

3. "SAVE" - клавиша "S" - сохранение файлов.

SAVE - сохраняет все загруженные файлы без пауз.

SAVE N - сохраняет все файлы, начиная с файла с номером N.

SAVE N TO NN - сохраняет файлы с номерами от N до NN.

SAVE TO N - сохраняет файлы с номерами от 1 до N.

SAVE STEP N - сохраняет все загруженные файлы, между файлами делает паузы в N секунд.

SAVE N TO NN STEP M - сохраняет файлы с номерами от N до NN, где N - номер первого файла; NN -номер последнего файла; M - пауза между файлами в секундах.

4. "VERIFY" клавиша "V" - проверка сохраненных файлов

VERIFY - аналогично SAVE.

VERIFY N TO NN - аналогично SAVE N TO NN.

VERIFY N аналогично SAVE N.

5. "LET" - клавиша "L" - изменение полей заголовка файла,

например:

```
LET 2=AAA,,1
```

Файл с номером 2 будет иметь имя AAA и стартовый адрес 1.

6."LIST" - клавиша "K" распечатка памяти.

LIST (NN) - задает адрес памяти (по умолчанию принимается равным 0) по этой команде выводится 15 байтов памяти, для каждого из которых приводятся:

- адрес памяти;
- десятичное значение байтов;
- десятичное значение двух смежных байтов;
- символьное значение байта.

Для продолжения вывода информации на экран, т.е. просмотра следующих 15 байтов нажмите ENTER.

7. "POKE" - клавиша "O" - изменение десятичного значения байта.

POKE x,NN

- x - адрес;
- NN - значение двух смежных байтов (>255),

POKE X, N

- X - адрес;
- N - десятичное значение байта (<256);

Примечание: Если значение N лежит в диапазоне 256-65535, то считается, что задано значение двух смежных байтов.

8. "USR" - клавиша "U" - вызывает подпрограмму пользователя.

USR X - вызывает подпрограмму в машинных кодах, расположенную по адресу X. Например, если вы желаете полностью перезапустить систему вашего компьютера, наберите USR 0.

9. "RETURN" - клавиша "7"

- возврат в МОНИТОР, инициализируются системные переменные и таблица каналов, однако полный сброс не выполняется.

10. "COPY" - клавиша "Z"

- осуществляет перевод программы в специальный режим для копирования файлов без заголовков длиной до 49056 байт.

После выдачи команды программа загружает файл в память, а затем, по нажатию клавиши "CAPS SHIFT" выгружает ее необходимое число раз. Повторная загрузка возможна только, если остается не менее 200 байт свободной памяти.

COPY NN - данная команда осуществляет копирование файлов, длиной до 49153 байта, копирование выполняется только один раз.

В заключение рассмотрения работы копировщика приводим условное обозначения типов файлов, используемых при его работе.

P - программа

B - вычислительный код (BYTES)

A - числовой массив

$ - символьный массив

2.2.2 Изменение хэдера для блокировки автозапуска.

В главе 1 мы с Вами достаточно подробно рассмотрели структуру хэдера. Напомним лишь, что именно в хэдере Бейсика задается параметр автостарта (т.е. его наличие или отсутствие и номер строки автозапуска при наличии такового). А поскольку мы используем в своей работе такое универсальное средство, как COPY-COPY, то можем изменить эти параметры вплоть до ликвидации автозапуска программы. Рассмотрим более подробно этот процесс.

Как Вам уже вероятно известно, байты 15 и 16 хэдера интерпретируются по-разному. В заголовках программ, написанных на Бейсике, эти байты содержат номер строки, с которой запускается программа - т.е. номер строки автостарта. Если же программа была записана без опции LINE и после считывания не запускается автоматически, то значение числа, содержащегося в этих двух байтах больше 32767. Как видим, одним из способов нейтрализации самозапускающихся программ, является замена этих двух байтов на число, большее 32767. Осуществить это нам поможет программа COPY-COPY.

Для этой цели загрузим копировщик и считаем необходимый нам заголовок с ленты. После этого и будем собственно осуществлять изменения. COPY-COPY настолько универсальная вещь, что изменения можно производить двумя способами. Первый способ основан на изменении встроенной функции LET, а второй использует оператор LIST для просмотра и POKE для непосредственного изменения содержимого ячеек памяти. Рассмотрим более подробно каждый из этих методов.

Метод первый.

Для того, чтобы использовать возможности функции LET, необходимо иметь представление, в каком формате она задается. Как Вам уже вероятно известно, данная функция в общем виде может быть представлена, как:

LET = имя программы, длина программы, номер строки автостарта, прочие

параметры.

Примечания: здесь рассмотрено применение функции LET копировщика COPY-COPY для изменения параметров Бейсик-хэдера. Для других типов хэдера общий вид будет несколько иным.

Подобная структура общего вида команды LET говорит о том, что если мы хотим изменить параметры хэдера следующего в списке копировщика под номером N, то мы должны набрать соответствующую команду LET, после которой через запятую набирается имя программы, длина программы, номер строки автостарта и т.д. Если же мы не хотим изменять все параметры хэдера, то нам необходимо соблюдать прежний порядок набора команды, только вместо параметров, которые мы желаем оставить неизменными, ничего не набираем, оставляя при этом необходимый контингент запятых, например: (исходный хэдер загружен под номером 1 и мы желаем только исключить автозапуск программы)

```
LET 1=..32768
```

После того, как Вы введете эту команду в копировщике COPY-COPY, то исходный хэдер не будет автоматически запускать исходную программу.

Метод второй.

Основан на получении дампинга хэдера путем встроенных функций COPY-COPY с последующим изменением содержимого ячеек памяти.

Для начала получим дампинг хэдера в том формате, который выдает копировщик.

Для того, чтобы вам было легко сориентироваться, напомним, что после 10 байтов названия идут 2 байта длины блока программы, после которых следует 2 байта, которые характеризуют автозапуск программы на Бейсике, именно эти байты и необходимо изменять для того, чтобы выполнить поставленную задачу. Следует помнить, что в этой паре байтов сначала идет старший байт, а потом младший.

Необходимо напомнить читателю, что для получения дампинга необходимо подать команду LIST 23296. (Если исходный хэдер идет первым - в противном случае необходимо сделать так, чтобы он шел первым).

Для изменения содержимого ячеек памяти необходимо использовать встроенную команду POKE. Ее использование полностью аналогично использованию данной команды в Бейсике ZX SPECTRUM.

Данные два метода изменений в программе COPY-COPY полностью альтернативны и взаимозаменяемы. Первый метод несколько более прост в использовании, однако применение второго метода позволяет вам непосредственно изучить структуру хэдера, что бывает иногда необходимо при детальном исследовании какой-либо конкретной программы.

После того, как Вам удалось изменить хэдер и создать необходимую конфигурацию, блокирующую автозапуск, необходимо выгрузить заголовок на магнитную ленту. Делается это с помощью опции копировщика SAVE.

Теперь, если Вы загрузите Вашу исходную программу вместе с измененным хэдером, то Вам удастся достаточно легко изучить ее структуру данной программы.

Приведенный в этой главе метод взлома является несколько более совершенным, в сравнении с методом, описанным в разделе 2.2.1. Однако, он все еще имеет ряд недостатков, и в первую очередь наиболее неприятным является тот факт, что Вам все еще приходится использовать место на кассете. А во-вторых, не все пользователи имеют копировщик COPY-COPY.

Устранению всех этих недостатков способствует использование при взломе специальной программы, которая будет описана в следующей статье.

2.3 Универсальный метод взлома с использованием специального программного обеспечения.

Как Вы уже вероятно догадались, использование методов описанных в этой главе, сопряжено с некоторыми сложностями и неудобствами. Одним из наиболее существенных является необходимость манипуляций с магнитофоном для записи промежуточных файлов.

Метод, который описан в этом разделе, лишен этого и других недостатков.

Он основан на использовании специальной программы, благодаря которой нам удастся блокировать автозапуск исходной программы. Если ввести в память компьютера эту программу и запустить ее, то после запуска программа начинает ждать первую программу на Бейсике, находящуюся на ленте. Она считывает ее аналогично команде LOAD, однако после загрузки не позволяет программе запуститься - выводит сообщение "0 OK". Кроме этого, данная программа выводит информацию, с какой строки считанная программа должна стартовать.

Теперь, когда Вы получили всю необходимую информацию, можно достаточно быстро изучить структуру Бейсик-файла исходной программы.

Универсальная программа для блокировки автозапуска.

```
1 FOR i=60000 TO 60025: READ A: POKE N,A: 2 NEXT i
3 RANDOMIZE USR 60000
4 DATA 1, 34, 0, 247, 213, 221, 225, 253, 54, 56, 1, 221,54, 1,225, 205,29,7,42, 66, 92, 34,
      69, 92, 207, 255
```

Ниже приведен дисассемблер программы в кодах, которая формируется в данном случае с использованием блока DATA.

```
10    ORG 60000
20    LD BC,34
30    RST 48
40    PUSH DE
50    POP IX
60    LD (IX+58),1
70    LD (IX+1),255
80    CALL 1821
90    LD HL,(23618)
100   LD (23621),HL
110   RST 08
120   DEFB 255
```

Глава 3. Методика просмотра Бейсик - программ.

3.1 Просмотр строк, защищенных управляющими кодами.

Информации, которою читатель получил, ознакомившись с предыдущими главами, достаточно для блокировки автозапуска любой Бейсик-программы к ZX SPECTRUM. Но, преодолев первичную защиту, Вы сталкиваетесь со вторым барьером, порой куда более сложным - защитные управляющие коды (подробно рассмотрены в главе 2 т. 1), а также со встроенными процедурами в машинных кодах (рассмотрены в первой главе т. 1).

Введение в компьютерную программу управляющих кодов может преследовать самые разнообразные цели. Одной из них является создание красочной цветовой гаммы и оптимального расположения информации на экране компьютера при минимальном количестве расходуемой памяти. Это используется в некоторых Бейсик-программах, однако, наибольшее распространение управляющие коды получили в системах защиты, где они кроме вышеописанных свойств приобретают еще функции элементов, препятствующих просмотру содержимого программы рядовым пользователем. Здесь мы рассмотрим некоторые ситуации, возникающие при исследовании программ, содержащих управляющие символы, а также методику блокировки управляющих кодов защиты.

Наиболее часто в компьютерных программах к ZX SPECTRUM используются:

CHR$ 8 BACKSPACE

CHR$ 16 INK CONTROL

CHR$ 17 PAPER CONTROL

Именно методику блокировки этих управляющих кодов мы с Вами и рассмотрим.

Для начала коротко об аспектах применения данных управляющих кодов.

BACKSPACE - "курсор влево" служит для забивания предыдущего символа. Именно этот управляющий код генерируется специальными процедурами компьютера для перевода

курсора влево. В защите программ он имеет несколько иное назначение! Благодаря ему удается скрывать ключевые слова, символы, а также элементы слов. В основном это используется для придания эффекта "солидной защиты" и достаточно редко применение данного управляющего кода связано с дезинформацией. По части введения в заблуждение, а также сокрытия наиболее уязвимой программной информации наибольшее распространение получили управляющие коды INC CONTROL и PAPER CONTROL. Именно их применение во многих случаях скрывает от вас подлинный текст программы, поэтому знание принципов блокировки этих управляющих кодов просто "жизненно необходимо".

При разработке технологии применения блокировки данного типа защиты я руководствовался интересами пользователя. В самом деле, почему пользователь должен что-либо делать, если эти функции можно возложить на компьютер.

Это привело к созданию универсальной программы, применение которой позволит избавить текст исходной программы от управляющих символов. Эта программа является более расширенной по своим возможностям в сравнении с аналогичной программой, рассмотренной в главе 1.

Ниже приведен ее листинг.

```
9990 REM ПРОГРАММИСТ МИХАЙЛЕНКО ВАДИМ МЕНСК МРТИ 1991
9991 PAPER 7:INK 0: BORDER 7:CLS
9992 FOR i=23758 TO 65000
9993 IF PEEK i = 13 THEN IF PEEK (i+1) = 39 AND PEEK(i+2)=6 THEN LIST: STOP
9994 IF PEEK i = 13 THEN LET i=i+4
9995 IF PEEK i = 16 THEN POKE(i+1),0: LET i=i + 2
9996 IF PEEK i = 17 THEN POKE(i+1),7: LET i=i+2
9997 IF PEEK i = 8 THEN POKE i,32 9998 NEXT i
```

Примечание ИНФОРКОМа:

У нас есть небольшое замечание, которое мы при редактировании не внесли в листинг, поскольку он защищен авторской строкой 9990.

Суть его в том, что коды 15, 17, 13, 8 и др. могут появляться в БЕЙСИК-строках и не быть управляющими кодами. Вы знаете, что в БЕЙСИКе после обычного посимвольного представления чисел идет код CHR 14 (NUMBER), после которого то же число представляется в скрытой пятибайтной форме (интегральная форма действительных чисел). Так вот, в этих пяти байтах могут быть любые числа, в том числе и те, которые программа может принять за управляющий код. Этот случай, в принципе надо обходить, например добавив в строке 9994 в ее конце после двоеточия:

```
: IF PEEK i=14 THEN LET i=i+5
```

Отметим, что это упущение никак не влияет на получение листинга со снятыми управляющими кодами, и мы упоминаем об этом только в образовательных целях.

Действует программа следующим образом. После того, как были установлены цвета символов, фона и бордюра, в цикле идет анализ Бейсик-строк. Строка 9993 следит за тем, чтобы не обрабатывалась программа блокировки управляющих кодов. Здесь фиксируется конец предыдущей строки Бейсика и проверяется, не имеет ли следующая строка номер 9990 (Вот почему наличие строки с таким номером обязательно). После того, как эта строка обнаружена (следовательно, вся предыдущая Бейсик-программа уже подверглась обработке) - программа блокировки распечатывает на экране текст исходной программы таким, каким он предстает без управляющих кодов. После того, как это сделано, программа останавливается оператором STOP.

Строка с номером 9994 имеет двойное предназначение. Т.к. всякая строка Бейсика оканчивается кодом ENTER - 0BH, то следовательно мы можем определить окончание Бейсик-строки. Кроме того, известно, что первые 4 символа в строке - это соответствующее кодовое представление номера и длины строки. Номер строки нашей программы может содержать одним из кодов число, равное искомому управляющему коду, но поскольку Бейсик-интерпретатором данная последовательность обрабатывается именно как номер и длина, то изменение значений данных ячеек памяти было бы ошибкой. Поэтому мы не анализируем данные 4 байта. Кроме этого, данный алгоритм несколько ускоряет работу

программы.

Строки 9995-9997 осуществляют поиск управляющих кодов и осуществляют все необходимые изменения. В соответствии с установленными в начале программы (см. строка 9991) значениями цвета осуществляется принудительная установка INK CONTROL в черный цвет, а PAPER CONTROL в белый.

В случае же обнаружения управляющего кода BACKSPACE осуществляется принудительная замена его на код пробела - 32.

Несмотря на всю свою привлекательность, основой которой является доступность для понимания, она обладает рядом существенных недостатков, которые затрудняют ее применение на практике. В частности, это низкое быстродействие и сложность внедрения данной программы-резидента в некоторые типы исходных программ, с которых необходимо снять защиту (бывает, что строки с номерами программы-резидента уже задействованы в исходной программе). Кроме этого, данный резидент проверяет не все типы управляющих кодов (а в качестве защиты от листинга может быть использован практически любой из них). Если же мы дополним резидент еще рядом операций по обезвреживанию всех управляющих кодов, то ее объем значительно увеличится, что повлечет за собой увеличение времени работы в несколько раз. Наиболее радикальным средством для увеличения быстродействия является программирование в машинных кодах, именно в этой области вы сможете получить максимальную скорость работы компьютера. К тому же, при специальной системе программирования эти типы программ неприхотливы к месту свободной оперативной памяти, в котором их размещают. Поэтому, если у Вас имеется возможность, старайтесь всегда переводить свои алгоритмы на язык Ассемблера. Это не только ускорит работу Ваших программ на порядок, но еще и поможет вам на практике изучить действительные принципы работы компьютера (процессора Z80).

Вашему вниманию предлагается программа блокировки защиты из управляющих кодов, написанная на языке Ассемблера. Разумеется, я понимаю, что не все читатели знакомы с данным языком программирования, поэтому ниже описаны принципы ее работы достаточно детально. (Тем, кто не знает, как приступить к подобному типу программ, рекомендую трехтомник "Первые шаги в машинных кодах" - ИНФОРКОМ, 1990).

```
Programming by Mihailenko Vadim.
All rights reserved. Mensk 1991.
Mihailenko Vadim driver system
for "EDITAS-48" files. Special
for "INFORCOM" corporation.

10                ORG 62030
20 ;              -------------------
30 ;              PARAMETR
40                LD HL,200
50                LD BC, (23635)
60 ;              -------------------
70 ;              BREAKCONTROL
60    VOZVR       CALL 8020
90                JR NC,BREAK
100               LD A, (BC)
110 ;             -------------------
120 ;             ANALIZ
130               CP 13
140               JR Z,STROK
150               CP 16
160               JR Z,INK
170               CP 17
180               JR Z,PAPER
190               CP 8
200               JR Z,BACK
210 ;             -------------------
220 ;             BLOKIROVKA
230 ;             KODOV
```

```
240               CP 18
250               JR Z,BACK
260               CP 19
270               JR Z,BACK
260               CP 20
290               JR Z,BACK
300               CP 21
310               JR Z,BACK
320               CP 22
330               JR Z,BACK
340               CP 23
350               JR Z,BACK
360 ;             -------------------------
370 ;             NEXT PARAMETR
380               DEC HL
390               INC BC
400 ;             -------------------------
410 ;             ENDCONTROL
420               LD A,H
430               CP 0
440               JR Z,ZERO
450               JR VOZVR
460 ;             -------------------------
470 ;             SUBROUTINES
480 ;             END CONTROL
490 ;             ZERO L
500   ZERO        LD A,L
510               CP 0
520               RET Z
530               JR VOZVR
540 ;             -------------------------
550 ;             13TH CONTROL
560   STROK       DEC HL
570               DEC HL
580               DEC HL
590               DEC HL
600               INC BC
610               INC BC
620               INC BC
630               INC BC
640               JR VOZVR
650 ;             ----------------------
660 ;             INK CONTROL
670   INK         DEC HL
690               LD A, 0
700               LD (BC),A
710               JR VOZVR
720 ;             ----------------------
750 ;             PAP.CONTROL
740   PAPER       DEC HL
750               INC BC
760               LD A,7
770               LD (BC),A
780               JR VOZVR
790 ;             ----------------------
800 ;             CODE - 32
810               BACK LD A,32
820               LD (BC),A
630               JR VOZVR
840 ;             ----------------------
850 ;             WHEN BREAK
660 ;             THEN RESTART
870   BREAK       RST 8
880               DEFB 20
890               END
```

Данная подпрограмма в машинных кодах представляет собой универсальный инструмент для снятия защиты, состоящей из управляющих кодов. Эта программа является логическим продолжением программы для блокировки автозапуска (описана в третьем разделе предыдущей главы). Вместе они представляют собой универсальное средство для просмотра содержания любого типа Бейсик-программ. (Методика объединения данных программ будет описана ниже). Кроме того, как самостоятельная программная единица, она позволяет размещать себя в любом месте оперативной памяти ZX SPECTRUM. Это намного расширяет спектр ее возможных применений.

Приведенная ранее Бейсик-программа поможет Вам лучше понять алгоритм. Фактически - это более модернизированный аналог.

Предполагается, что эта программа будет сформирована с адреса 62030, оставляя предыдущие 25 байтов для программы блокировки автозапуска с целью совместного применения.

Первый блок PARAMETR задает параметры работы программы. Здесь в регистр HL заносится длина обрабатываемой программы (в моем случае она равна 200, но это очень легко можно будет изменить, о том, как это осуществить, будет описано ниже). В регистр BC заносится содержимое системной переменной PROG. Это необходимо для того, чтобы определить начальную точку работы программы. Поскольку мы собирается корректировать содержимое Бейсик-файла, то нам естественно необходимо знать адрес, с которого он начинается, а именно на него и указывает содержимое переменной PROG.

Следующий блок программы - это анализ в цикле содержимого текущей ячейки памяти, которое заносится в аккумулятор. Это очень похоже на принцип работы Бейсик-программы аналогичного назначения. Фактически эта часть программы, работающая в цикле, состоит из пяти небольших блоков: BREAKCONTROL, ANALIZ, BLOKIROVKA KODOV, NEXT PARAMETRES и ENDCONTROL.

BREAKCONTROL осуществляет проверку нажатия клавиши BREAK. Для этой цели используется встроенная процедура, вызов которой осуществляется командой CALL. В случае же нажатия клавиши BREAK (это определяется по состоянию флага C регистра F) осуществляется переход на подпрограмму BREAK, которая осуществляет рестарт с выдачей сообщения о нажатии клавиши BREAK.

В этом же блоке осуществляется загрузка в аккумулятор текущего содержимого ячейки памяти, адрес которой определяется содержимым регистра BC.

Блок ANALIZ ведет последовательную проверку содержимого аккумулятора, используя функцию сравнения Z80 – CP. Наиболее приоритетным здесь является проверка наличия кода ENTER, поэтому этот параметр проверяется в первую очередь. В случае обнаружения этого кода осуществляется переход на подпрограмму 13TH CONTROL, которая увеличивает содержимое регистра BC на 4 для того, чтобы анализу не подверглись номер и длина строки Бейсика. Параллельно с увеличением на 4 регистра BC, происходит уменьшение регистра HL, который служит счетчиком, и по которому определяется окончание работы программы.

Если же содержимое ячейки не является кодом ENTER, то осуществляется проверка его на код INK CONTROL и PAPER CONTROL. В случае, если оно соответствует какому-либо из этих значений, то осуществляется перевод на подпрограмму обработки INC_CONTROL или PAP_CONTROL соответственно.

Эти подпрограммы осуществляют принудительный ввод кодов черного и белого цвета, аналогично тому, как это осуществила Бейсик-программа.

Следующей идет проверка наличия кода BACKSPACE. Соответствующая подпрограмма осуществляет замену этого кода на код пробела 32, поскольку в большинстве случаев BACKSPACE используется для скрытия каких-то определенных элементов программы, которые, однако оказываются доступны просмотру при замене его на код SPACE. Фактически, если бы мы ограничились только блоком ANALIZ, то мы имели бы полный аналог описанной выше программы на Бейсике. Однако, поскольку набор управляющих кодов не ограничивается лишь только INK CONTR, PAPER CONTROL и

BACKSPACE, а существуют еще OVER CONTROL, BRIGHT CONTROL, INVERSE CONTROL, FLASH CONTROL, AT и TAB CONTROL (а для защиты от листинга может с успехом использоваться практически любой из них), то программа в машинных кодах имеет расширение, осуществляющее контроль наличия всех вышеописанных управляющих кодов. В случае их обнаружения осуществляется замена их кодом пробела - 32 с использованием подпрограммы BACK. Блок NEXT PARAMETRS изменяет содержимое контрольных регистров HL и BC таким образом, чтобы осуществлялся анализ следующих ячеек памяти.

Блок END CONTROL осуществляет контроль окончания программы по содержимому регистра HL. Если счетчик HL содержит 0, то осуществляется возврат в вызывающую программу по команде SET. Контроль осуществляется следующим образом.

Сначала проверяется содержимое старшего разряда регистра HL и сравнивается с 0. Если оно равно 0, то осуществляется проверка содержимого младшего разряда данного регистра. Если и оно равно 0, то осуществляется выход и программа продолжает анализ содержимого текущих ячеек оперативной памяти компьютера.

Мы с Вами рассмотрели принцип работы программы блокировки управляющих кодов. Теперь рассмотрим некоторые аспекты ее практического применения.

Как уже было отмечено выше, данная программа может работать в любом свободном месте оперативной памяти. Это достигается за счет использования команд относительного перехода JR (ввиду того, что объем программы незначителен, применение функции JR вполне допустимо).

Как было подчеркнуто, данная программа допускает совместное использование с программой автозапуска. Это допустимо потому, что программа блокировки автозапуска формируется с адреса 62000 и занимает 25 байтов. С учетом этого, можно записать общий блок кодов как самостоятельную программную единицу, подав команду:

```
SAVE "BLOKIR" CODE 62000, 135
```

Теперь вы можете, загрузив этот блок, вызывать данные процедуры, давая команды:

```
RANDOMIZE USR 62000
```

для процедуры блокировки автозапуска

и

```
RANDOMIZE USR 62030
```

для блокировки управляющих кодов.

Примечание. Необходимо отметить, что перед тем, как выгрузить данный блок кодов на магнитофон, необходимо сформировать программу блокировки автозапуска, начиная с адреса 62000, что достигается путем использования программы на Бейсике, описанной в 2.2.3. Для того, чтобы сформировать процедуру обработки управляющих кодов можно тоже воспользоваться Бейсиком, используя десятичную последовательность кодов как блок DATA.

Для этого можно использовать достаточно простую программу.

```
10 FOR I=63030 TO 62130
20 READ N: POKE I, N
30 NEXT I
40 DATA 33,200,0,237,75,83,92,205
```

и т.д. в соответствии с приведенными ниже значениями:

```
Programming by Mihailenko Vadim. All rights reserved. Mensk1991.
Mihailenko Vadim driver system for "EDITAS-48" files.
Special for "INFORCOM" Corporation.

33,   200,  0,    237,  75,   83,   92,   205,
84,   31,   48,   85,
10,   254,  13,   40,
51,   254,  16,   40,   57,   254,  16,   40,
60,   254,  8,    40,
63,   254,  18,   40,
59,   254,  19,   40,
```

```
55,    254,  20,    40,
51,    254,  21,    40,
47,    254,  22,    40,
43,    254,  23,    40,
39,    43,   3,     124,
254,   0,    40,    2,
24,    201,  125,   254,
0,     200,  24,    195,
43,    43,   43,    43
3,     3,    3,     3,
24,    185,  43,    3,
62,    0,    2,     24,
178,   43,   3,     62,
7,     2,    24,    171,
62,    32,   2,     24,
166,   207,  20,    0,
```

Если же вы желаете сформировать данную последовательность кодов в другом месте оперативной памяти, то необходимо произвести соответствующие изменения в строке 10 Бейсик-программы, указав вместо 62030 необходимое значение.

Теперь рассмотрим небольшую особенность данной программы, связанную с длиной обрабатываемой Бейсик-программы, т.е. с длиной области оперативной памяти, которая очищается от защитных управляющих кодов. В моем варианте мы обрабатываем 200 байтов оперативной памяти, однако бывают случаи, когда этого оказывается недостаточно. Чтобы увеличить обрабатываемую область, необходимо увеличить число, заносимое в регистр HL. Если Вы сформировали данную программу с адреса 62030, то значение, заносимое в HL будет характеризоваться двумя байтами 62031 и 62032, причем сначала идет младший байт, а потом старший. Если Вы хотите сделать величину обрабатываемой области X, то Вам необходимо ввести соответствующую строку с клавиатуры:

```
LET A=INT(X/255): POKE 62032,A; LET B=X-A*255: POKE 62031,B
```

Если же Вы хотите подбирать значение переменной X, то можно оформить эту последовательность операндов, как строку Бейсик-программы.

Следует отметить еще одну немаловажную деталь, характеризующую работу данной программы. Ввиду того, что достаточно часто для защиты используются встроенные процедуры в машинных кодах (они непроизвольно останавливают листинг с выдачей сообщения INVALID COLOR или NUMBER TOO BIG) совместно с методом зануления, а данная программа блокировки управляющих кодов не делает различие между строками Бейсика и встроенными процедурами, то происходит их полная переработка (анализ), что, с одной стороны, позволяет Вам получить полный листинг программы, но с другой стороны, ввиду изменения во встроенных процедурах машинных кодов мы не можем вызывать эти процедуры сразу после просмотра листинга, чтобы увидеть весь эффект их действия. Для того, чтобы все же наблюдать данный эффект, необходимо вновь загрузить исходный Бейсик-файл и осуществлять проверку до использования программы блокировки управляющих кодов.

Примечание ИНФОРКОМа. Мы не сможем запускать программу со снятыми кодами и по причине отмеченной выше - в связи с тем, что коррумпируются числа в БЕЙСИК-строках.

Внимание!

Данная машиннокодовая программа для блокировки действия управляющих кодов написана Михайленко Вадимом. При использовании в разработках указывать автора.

(Продолжение следует)

Дж. Хардман, Э. Хьюзон
Перевод Тихоновой Н.А.

40 ЛУЧШИХ ПРОЦЕДУР

Окончание.
Начало см. с. 17-28, 61-70, 105-110.

8.5 Составление списка переменных.

Длина: 94
Количество переменных: 0
Контрольная сумма: 10295
Назначение:
Эта подпрограмма составляет список имен всех переменных, имевшихся в настоящий момент в памяти.
Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:
Если переменных в памяти нет, программа возвращается в BASIC.
Комментарий:
Это большая помощь при отладке программ, особенно длинных и сложных.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
METKA       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
            RES 0,(IY+2)       253   203   2     134
            LD HL,(23627)      42    75    92
N_VAR       LD A,              13    62    13
            RST 16             215
            LD A,              32    62    32
            RST 16             215
            LD A,(HL)          126
            CP 128             254   128
            RET Z              200
            BIT 7,A            203   127
            JR Z,BIT_5         40    62
            BIT 6,A            203   119
            JR Z,N_BIT         40    31
            BIT 5,A            203   111
            JR Z,STR_AR        40    9
            SUB 128            214   128
            LD DE,19           17    19    0
PRINT       RST 16             215
            ADD HL,DE          25
            JR N_VAR           24    225
STR_AR      SUB 96             214   96
            RST 16             215
            LD A, 36           62    36
BRACK       RST 16             215
            LD A,40            62    4     0
            RST 16             215
            LD A,41            62    41
POINT       INC HL             35
            LD E,(HL)          94
            INC HL             35
            LD D,(HL)          86
            INC HL             35
            JR PRINT           24    234
```

```
N_BIT       BIT 5,A           203   111
            JR Z,ARRAY        40    19
            SUB 64            214   64
            RST 16            215
NEXT_C      INC HL            35
            LD A,(HL)         126
            BIT 7,A           203   127
            JR NZ,LAST_C      32    3
            RST 16            215
            JR NEXT_C         24    247
LAST_C      SUB 128           214   128
JUMP        LD DE,6           17    6     0
            JR PRINT          24    211
ARRAY       SUB 32            214   32
            JR BRACK          24    216
BIT_5       BIT 5,A           203   111
            JR NZ,JUMP        32    243
            ADD A,32          198   32
            RST 16            215
            LD A, 36          62    36
            JR POINT          24    211
```

Как она работает:

Бит 0 байта по адресу 23612 сбрасывается, чтобы символы, выводимые на печать, появлялись в верхней части экрана. В HL загружается адрес области переменных. В аккумулятор загружается признак ENTER и вызывается подпрограмма ПЗУ по адресу 16. В аккумулятор затем загружается код пробела, и вновь вызывается та же самая подпрограмма ПЗУ.

В аккумулятор загружается байт по адресу в HL. Если значение этого байта равно 128, программа возвращается в BASIC, т.к. достигнут конец области переменных.

Если бит 7 аккумулятора установлен в 0, программа переходит к 'BIT_5', т.к. встретились строковая переменная или число, имя которых состоит только из одной буквы. Проверяется 6 бит аккумулятора. Если он равен 0, делается переход к 'N_BIT', т.к. определены массив или число, имя которых более, чем одна буква. Если бит 5 аккумулятора равен 0, программа переходит к 'STR_AR'

Программа достигает этой точки, если найденная переменная является управляющей переменной цикла FOR/NEXT. В этом случае из аккумулятора вычитается 128 и результатом является код символа для вывода на печать. В пару DE загружается число 19, указывая на следующую переменную при прибавлении к HL. Символ в аккумуляторе выводится на печать, DE прибавляется к HL, а программа возвращается к поиску следующей переменной 'N_VAR'.

Если программа находит строковый массив (достигает 'STR_AR'), то из аккумулятора вычитается число 96, что дает код имени найденного массива. Это значение выводится на печать, используя подпрограмму ПЗУ. Знак доллара и левая скобка выводятся на печать, а в аккумулятор загружается код правой скобки. HL увеличивается, указывая на байты, содержащие длину массива. Это значение загружается в DE, так что прибавление к HL дает адрес следующей переменной. Делается переход к 'PRINT', где правая скобка выводятся на печать, и DE прибавляется к HL.

В процедуре 'N_BIT' проверяется бит 5 аккумулятора. Если он установлен в 0, т.е. это числовой массив, то происходит переход к 'ARRAY'. Если он установлен в 1, то это числовая переменная, имя которой длиннее, чем одна литера. Из аккумулятора вычитается 64, а полученный в результате символ выводится на печать. Затем программа выполняет цикл, выводя на печать каждый встретившийся символ, до тех пор, пока находится хоть один символ с битом 7, установленным в 1. Из кода этого последнего символа вычитается 128, в DE загружается смещение для следующей переменной, а программа переходит к 'PRINT'.

Если массив найден, 32 вычитается из аккумулятора, чтобы получить правильный код, и делается переход на поиск скобок ' BRACK'.

В процедуре 'BIT_5', если найдено число, имя которого имеет только одну букву,

программа возвращается к 'JUMP'.

Окончание подпрограммы работает, когда встречающиеся переменные - строковые. Прибавление 32 к аккумулятору дает код для вывода на печать. Наконец, в аккумулятор загружается код знака доллара и делается переход к 'POINT'

8.6 Поиск строки.

Длина: 155
Количество переменных: 2
Контрольная сумма: 17221
Назначение:

Эта программа осуществляет поиск по БЕЙСИК программе и выводит каждую строку, содержащую набор символов, определенных пользователем.

Переменные:
Имя - data start
Длина - 2
Адрес 23296
Комментарий: адрес первого байта данных.
Имя - string length
Длина - 1
Адрес - 23298
Комментарий: число символов в строке.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если в памяти нет БЕЙСИК-программы или символьная строка имеет нулевую длину - возврат в БЕЙСИК.

Комментарии:

Время выполнения этой программы пропорционально двум величинам: длине строковой переменной и длине БЕЙСИК программы. Строка для поиска должна быть помещена в ячейку выше RAMTOP, а адрес первого байта строки должен быть помещен в ячейки 23296/7. Длина строки должна быть сохранена в ячейке 23298.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
METKA       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
            RES 0,(IY+2)       253   203   2     134
            LD IX,(23296)      221   42    0     91
            LD HL,(23635)      42    83    92
RESTAR      LD A,(23298)       58    2     91
            LD E,A             95
            CP 0               254   0
            RET Z              200
            PUSH HL            229
RESTOR      PUSH IX            221   229
            POP BC             193
            LD D,              0     22    0
            INC HL             35
            INC HL             35
            INC HL             35
CHECK       INC HL             35
            PUSH DE            213
            LD DE,(23627)      237   91    75    92
            AND A              167
            SBC HL, DE         237   82
            ADD HL, DE         25
```

```
            POP DE              209
            JR C,ENTER          56      4
            POP HL              225
            RET                 201
LONG_J      JR RESTAR           24      223
ENTER       LD A, (HL)          126
            CP 13               254     13
            JR NZ,NUMBER        32      5
            INC HL              35
            POP BC              193
            PUSH HL             229
            JR RESTOR           24      221
NUMBER      CALL 6326           205     182     24
            JRNZ COMPAR         32      8
            DEC HL              43
DIFFER      PUSH IX             221     229
            POP BC              193
            LD D,               0       22      0
            JR CHECK            24      216
COMPAR      LD A,(BC)           10
            CP (HL)             190
            JR NZ,DIFFER        32      245
            INC BC              3
            INC D               20
            LD A, D             122
            CP E                1       87
            JR NZ,CHECK         32      206
            LD A,               13      62      1       3
            RST 16              215
            POP HL              225
            PUSH HL             229
            LD B,(HL)           70
            INC HL              35
            LD L,(HL)           110
            LD H,B              96
            LD DE,1000          17      232     3
            LD A,47             62      47
THOUS       INC A               60
            AND A               167
            SBC HL,DE           237     82
            JR NC,THOUS         48      250
            ADD HL,DE           25
            RST 16              215
            LD DE,100           17      100     0
            LD A,47             62      47
HUNDR       INC A               60
            AND A               157
            SBC HL,DE           237     82
            JR NC,HUNDR         48      250
            ADD HL,DE           25
            RST 16              215
            LD DE,10            17      10      0
            LD A,               47      62      47
TENS        INC A               60
            AND A               167
            SBC HL,DE           237     82
            JR NC,TENS          48      250
            ADD HL,DE           25
            RST 16              215
            LD A,L              125
            ADD A,48            198     48
            RST 16              215
            POP HL              225
            INC HL              35
            INC HL              35
```

```
            INC HL              35
NEXT_C      INC HL              35
            LD A,(HL)           126
LINEND      CP 13               254     13
            JR NZ,CHR_14        32      4
            RST 16              215
            INC HL              35
            JR LONG_J           24      155
CHR_14      CALL 6326           205     182     24
            JR Z,LINEND         40      243
            CP 32               254     32
            JR C,NEXT_C         56      237
            RST 16              215
            JR NEXT_C           24      234
```

Как она работает:

Бит 0 байта, хранящегося по адресу 23612 сбрасывается, чтобы символы, выводимые на печать, появлялись в верней части экрана. В IX загружается адрес первого байта данных. Это позволяет загрузить этот адрес в другую пару регистров, используя в меньшей степени буфер принтера. В HL, загружается адрес начала БЕЙСИК-программы.

В аккумулятор загружается длина эталонной строки, и это значение копируется в E-регистр. Если длина строки равна 0, программа возвращается в БЕЙСИК. Адрес в HL помещается в стек, храня положение искомой в настоящий момент строки в памяти.

Адрес данных копируется из IX в BC для большей доступности. В D-регистр загружается 0, т. е. количество найденных символов, равнозначных введенным данным. Пара регистров HL увеличивается на 3, указывая на старший байт указателя длины строки. HL увеличивается, указывая на следующий символ. Пара регистров DE сохраняется в стеке.

В DE загружается адрес области переменных, и это значение вычитается из HL. Если результат отрицательный, программа переходит к 'ENTER' после восстановления HL и возвращения из стека DE. Если результат был положительным, стек восстанавливается до своего первоначального размера и выполняется возврат в БЕЙСИК, т. к. достигнут конец БЕЙСИК-программы.

В процедуре 'ENTER' в аккумулятор загружается байт, хранящийся по адресу в HL. Если это не признак ENTER, происходит переход к 'NUMBER'. Если признак ENTER найден, HL увеличивается, указывая на начало следующей строки. Адрес предыдущей строки удаляется из стека и замещается новым значением в HL. Затем делается переход к 'RESTOR'. В процедуре 'NUMBER' вызывается подпрограмма ПЗУ, расположенная там по адресу 6326. Если символ в аккумуляторе является признаком NUMBER (CHR_14), HL увеличивается, указывая на первый символ после пятибайтного представления числа, определенного подпрограммой ПЗУ. Если признак NUMBER не обнаружен, программа переходит к 'COMPAR', иначе HL уменьшается и программа переводит к 'DIFFER'. BC копируется из IX, количество найденных символов сбрасывается в 0 и делается переход к 'CHECK' .

В процедуре 'COMPAR' в аккумулятор загружается байт, хранящийся по адресу в BC. Если это значение не то же самое, что и байт, хранящийся по адресу в HL, программа возвращается к 'DIFFER'.

BC увеличивается, указывая на следующий байт данных, и количество определенных символов увеличивается. Если это значение не равно длине символьной строки, программа возвращается к 'CHECK'. В аккумулятор загружается код признака ENTER, и это значение выводится на печать, используя команду RST 16. Адрес строки для вывода на печать загружается из стека в HL, номер строки затем копируется в HL через B-регистр. В DE загружается 1000 и в аккумулятор загружается значение, на 1 меньшее, чем код символа "0". Аккумулятор уменьшается, а DE повторно вычитается из HL до тех пор, пока HL не станет отрицательным. Затем DE прибавляется к HL, чтобы получить положительный остаток. Символ из аккумулятора выводится на печать.

Вышеописанный прием повторяется затем для DE=100 и DE=10. Затем остаток загружается в аккумулятор, прибавляется 48, и в результате полученный символ выводится

на печать.

Адрес начала строки восстанавливается из стека и загружается в HL. Затем HL увеличивается, указывая на старший байт указателя длины строки, HL увеличивается, и байт с адресом в HL загружается в аккумулятор. Если этот байт не является признаком ENTER, делается переход к 'CHR_14', иначе ENTER выводится на печать, HL увеличивается, и программа возвращается к 'RESTAR'.

В процедуре 'CHR_14' вызывается подпрограмма ПЗУ по адресу 6326. Если символ в аккумуляторе является признаком числа, HL увеличивается, указывая на первый символ, стоящий после найденного числа. Этот символ загружается в аккумулятор и делается переход к 'LINEND'. Затем, если символ в аккумуляторе имеет код меньший, чем 32, подпрограмма возвращается к 'NEXT_C'. Если код больше, чем 31, найденный символ выводится на печать и происходит переход к 'NEXT_C'.

8.7 Поиск и замещение строки.

Длина: 85

Количество переменных: 3

Контрольная сумма: 8518

Назначение:

Программа шлет символьную строку в БЕЙСИК-программе и делает замену каждой найденной строки стринга на другую строку такой же длины.

Переменные:

Имя - old data start

Длина - 2

Адрес - 23296

Комментарий: адрес первого байта замещаемой строки.

Имя - string length

длина - 1

Адрес - 23298

Комментарий: длина замещаемой строки.

Имя - new data start

Длина - 2

Адрес - 23299

Комментарий: адрес первого байта замещающей строки.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Если длина строки равна 0 или БЕЙСИК-программы в памяти нет, то процедура возвращается непосредственно в БЕЙСИК.

Комментарий:

Время выполнения этой программы зависит от длины строки и от длины БЕЙСИК-программы.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
METKA       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
            LD IX,(23296)      221   42    0     91
            LD HL,(23635)      42    83    92
            LD A,(23298)       58    2     91
            LD E,A             95
            CP 0               254   0
            RET Z              200
            DEC HL             43
NEWLIN      INC HL             35
```

```
            INC HL              35
            INC HL              35
            INC HL              35
            JR RESET            24      23
CHECK       INC HL              35
            PUSH DE             213
            LD DE,(23627)       237     91      75      92
            AND A               167
            SBC HL,DE           237     82
            ADD HL,DE           25
            POP DE              209
            RET NC              208
            LD A,(HL)           126
            CP 13               254     13
            JR Z,NEWLIN         40      233
            CALL 6326           205     182     24
            JR NZ, COMPAR       32      8
            DEC HL              43
RESET       PUSH IX             221     229
            POP BC              193
            LD D,0              22      0
            JR CHECK            24      226
COMPAR      LD A,(BC)           10
            CP (HL)             190
            JR NZ,RESET         32      245
            INC BC              3
            INC D               20
            LD A,D              122
            CP E                187
            JR NZ,CHECK         32      216
            PUSH HL             229
            LD D,0              22      0
            AND A               167
            SBC HL,DE           237     82
            LD D, E             83
            LD BC,(23299)       237     75      3       91
            INC D               20
NEXT_CH     INC HL              35
            DEC D               21
            JR Z, FINISH        40      5
            LD A,(BC)           10
            LD (HL),A           119
            INC BC              3
            JR NEXT_C           24      247
FINISH      POP HL              225
            JR RESET            24      215
```

Как она работает:

В IX загружается адрес замещающей строки. Это значение должно быть выше RAMTOP. В HL загружается адрес начала программной области, а в аккумулятор загружается длина строки, которая копируется в E-регистр для дальнейшего использования в программе. Если длина строки равна 0, программа возвращается в БЕЙСИК.

Устанавливается HL, указывая на старший байт следующего указателя БЕЙСИК-строки и делается переход к 'RESET'.

В процедуре 'CHECK' HL увеличивается, указывая на следующий символ. DE сохраняется в стеке и загружается адресом области переменных. Если HL не меньше, чем DE, конец программы достигнут, и после восстановления DE из стека программа возвращается в БЕЙСИК.

В аккумулятор загружается символ по адресу в HL, Если это значение является знаком ENTER, программа возвращается к 'NEWLIN'. Если аккумулятор не содержит знак NUMBER (символ 14), делается переход к 'COMPAR', иначе HL увеличивается на 5, так что HL указывает на пятый байт найденного числа.

В процедуре 'RESET' в BC загружается адрес строки для поиска. Регистр D устанавливается в 0 для хранения количества символов в строке, найденной к тому времени. Программа затем возвращается к 'CHECK'.

В процедуре 'COMPARE' в аккумулятор загружается символ строки, на который указывает пара регистров BC. Если это значение отличается от байта по адресу в HL, программа переходит к 'RESET'. BC увеличивается, указывая на следующий символ в строке, регистр D, счетчик, увеличивается. Если это значение не равно длине строки, программа возвращается к 'CHECK'.

Если строка найдена, HL сохраняется на стеке, так что программа начинает поиск для следующего случая с этого адреса. В DE загружается длина строки и это значение вычитается из HL, давая значение на единицу меньше, чем стартовый адрес. Длина строки затем загружается в D для использования ее в качестве счетчика. В BC загружается стартовый адрес замещающей строки, а регистр D увеличивается. Регистр HL увеличивается, указывая на следующую ячейку, а счетчик уменьшается.

Если счетчик равен 0, HL восстанавливается из стека и делается переход к 'RESET' для следующего случая. В аккумулятор загружается символ, на который указывает BC и это значение помещается в ячейку по адресу в HL. BC увеличивается, указывая на следующий символ, а программа возвращается к 'NEXT CHAR'.

8.8 Поиск подстроки.

Длина: 168

Количество переменных: 0

Контрольная сумма: 19875

Назначение:

Эта программа возвращает позицию подстроки (B$) в главной строке (A$) или 0 в случае ошибки.

Вызов программы:

```
LET P = USR адрес контроль ошибок:
```

Если строка не существует или если длина подстроки равна нулю или если длина подстроки больше, чем длина главной строки, программа возвращает значение 0. Если ошибки нет, но подстрока не найдена в главной строке, программа также дает 0.

Комментарий:

После выхода из программы в машинных кодах переменная P (может быть использована любая другая переменная) будет хранить искомое значение. Строки, к которым делается обращение, не могут быть массивами данных. Для изменения используемых строк числа, отмеченные звездочкой, должны быть изменены. 66* - это подстрока, 65* - главная строка. Для изменения эти числа необходимо заменить кодами требуемых символов. Например, если Вы хотите найти позицию H$ в G$, то соответственно надо будет ввести 71 (код G) и 72 (код H)

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА
            SUB A              151
            LD B, A            71
            LD B, A            79
            LD D, A            87
            LD E, A            95
            LD HL,(23627)      42    75    92
NEXT_V      LD A,(HL)          126
            CP 128             254   128
            JR Z,NOT_FD        40    95
            BIT 7, A           203   127
            JR NZ,FOR_NX       32    41
            CP 96              254   96
            JR NC,NUMBER       48    29
```

```
            CP 65               254   65*
            JR NZ,SUBSTR        32    2
            LD D, H             84
            LD E, L             93
SUBSTR      CP 66               254   66*
            JR NZ,CHECK         32    2
            LD B,H              68
            LD C,L              77
CHECK       LD A,D              122
            OR E                179
            JR Z,STRING         40    4
            LD A,B              120
            OR C                177
            JR NZ,ROUND         32    38
STRING      PUSH DE             213
            INC HL              35
            LD E,(HL)           94
            INC HL              35
            LD D,(HL)           86
ADD         ADD HL,DE           25
            POP DE              209
            JR INCRS            24    5
NUMBER      INC HL              35
            INC HL              35
            INC HL              35
            INC HL              35
            INC HL              35
INCRS       INC HL              35
            JR NEXT_V           24    206
FOR_NX      CP 224              254   224
            JR C, N_BIT         56    6
            PUSH DE             213
            LD DE,18            17    18    0
            JR ADD              24    234
N_BIT       BIT 5,A             203   111
            JR Z,STRING         40    225
NEXT_B      INC HL              35
            BIT 7,(HL)          203   126
            JR Z,NEXT_B         40    251
            JR NUMBER           24    227
FOUND       EX DE,HL            235
            INC HL              35
            INC HL              35
            PUSH HL             289
            PUSH HL             229
            INC BC              3
            PUSH BC             197
            LD A,(BC)           10
            LD E,A              95
            INC BC              3
            LD A,(BC)           10
            LD D,A              87
            OR E                179
            JR Z,ZERO           40    11
            PUSH DE             213
            LD A,(HL)           126
            DEC HL              43
            LD L,(HL)           110
            LD H,A              103
            AND A               167
            SBC HL,DE           237   82
            JR NC,CONTIN        48    8
            POP BC              193
ZERO        POP BC              193
            POP BC              193
```

```
ERROR       POP BC              193
NOT_FD      LD BC,0             10    0
            RET                 201
CONTIN      POP IX              221   285
            POP BC              193
            EX DE,HL            235
            POP HL              225
            INC BC              3
            INC BC              3
SAVE        INC HL              35
            PUSH HL             229
            PUSH BC             197
            PUSH IX             221   229
            PUSH DE             213
COMPAR      LD A,(BC)           10
            CP (HL)             190
            JR Z,MATCH          40    12
            POP DE              209
            POP IX              221   225
            POP BC              193
            POP HL              225
            LD A,D              122
            OR E                179
            JR Z,ERROR          40    225
            DEC DE              27
            JR SAVE             24    234
MATCH       INC HL              35
            INC BC              3
            PUSH HL             229
            DEC IX              221   43
            PUSH IX             221   229
            POP HL              225
            LD A,H              124
            OR L                181
            POP HL              225
            JR NZ,COMPAR        32    227
            POP DE              209
            POP DE              209
            AND A               167
            SBC HL,DE           237   82
            POP DE              209
            POP DE              209
            POP DE              209
            AND A               167
            SBC HL, DE          237   82
            LD B,H              68
            LD C,L              77
            RET                 201
```

Как она работает:

В аккумулятор, пару регистров BC и пару регистров DE загружается 0. Позднее в программе в BC будет установлен адрес B$, а в DE будет установлен адрес A$. В HL загружается адрес начала области программных переменных.

В аккумулятор загружается байт из адреса, находящегося в HL. Если аккумулятор содержит число 128 программа переходит к 'NOT_FD', т.к. достигнут конец области программных переменных. Если бит 7 аккумулятора установлен в 1, делается переход к 'FOR_NX', так как найденная переменная - не строковая и не число, имя которого состоит только из одной литеры. Если аккумулятор содержит число большее, чем 95, делается переход к 'NUMBER'.

Для достижения этого этапа строка должна быть найдена. Если в аккумуляторе содержится число 65, определяется местонахождение строки A$, а содержимое HL копируется в DE. Если аккумулятор содержит число 66, определяется строка B$, а HL

копируется в BC. Если DE не равно 0, и BC не равно 0, определяется местонахождение обеих строк, и программа переходит к 'FOUND'.

Если программа достигает процедуры 'STRING', DE сохраняется в стеке и загружается длиной найденной строки. Это значение прибавляется к адресу старшего байта указателей строки и сохраняется в HL.. DE восстанавливается из стека и делается переход к 'INCRS'.

В процедуре 'NUMBER' HL увеличивается в пять раз, указывая на последний байт найденного числа. HL затем увеличивается, указывая на следующую переменную, и происходит переход к 'NEXT_V'.

В процедуре 'FOR_NX', если аккумулятор содержит число меньше, чем 224, делается переход к 'N_BIT', т.к. встретившаяся переменная не является управляющей переменной цикла FOR-NEXT. Если значение аккумулятора больше, чем 223, то число 18 прибавляется к HL, указывая на последний байт переменной цикла и программа возвращается к 'INCRS'.

Если программа достигает 'N_BIT', и бит 5 аккумулятора установлен в 0, делается переход к 'STRING', чтобы загрузить в HL адрес следующей переменной, т.к. найден массив.

Если программа достигает 'NEXT_B', найдено число, имя которого больше одного символа по длине. Т.о., HL увеличивается до тех пор, пока не укажет на последний символ имени переменной, а затем делается переход к 'NUMBER'.

В процедуре 'FOUND' в HL загружается адрес строки A$, и это значение увеличивается дважды, чтобы получить адрес старшего байта указателей. Это значение затем сохраняется в стеке дважды. BC увеличивается, указывая на младший байт указателей подстроки B$. Адрес в BC затеи сохраняется в стеке, в DE загружается длина строки B$, и, если это значение равно 0, делается переход к 'ZERO'. Затем DE помещается в стек. В HL загружается длина строки a$, и, если это значение не меньше, чем DE, программа переходит к 'CONTIN'. Указатель стека затем восстанавливается, в BC загружается 0, и программа возвращается в БЕЙСИК.

В процедуре 'CONTIN' в IX устанавливается длина строки B$, а в BC помещается адрес младшего байта указателей для подстроки B$. В DE загружается разность длин строк A$ и B$, а в HL загружается адрес старшего байта указателей для A$. BC затем увеличивается дважды, чтобы получить адрес первого символа в подстроке B$. HL увеличивается, указывая на следующий символ строки A$. HL, BC, IX и DE затем сохраняются на стеке. В аккумулятор загружается байт по адресу в BC, и, если это значение равно значению байта по адресу в HL, делается переход к 'MATCH'. DE, IX, BC и HL затем восстанавливаются из стека. Если DE содержит 0, делается переход к 'ERROR', т.к. подстроки B$ нет в строке A$. Счетчик DE уменьшается, и программа возвращается к 'SAVE'.

Если программа достигает процедуры 'MATCH', HL и BC увеличиваются, указывая на следующий символ A$ и B$ соответственно. HL затем сохраняется в стеке. IX, счетчик, уменьшается и после восстановления HL из стека, если IX не содержит 0, программа возвращается к 'COMPAR'.

Для достижения этого этапа местонахождение подстроки B$ в строке A$ уже должно быть определено. Длина подстроки B$ вычитается из HL, а затем адрес старшего байта указателей для строки A$ вычитается из HL. Результат - это позиция подстроки B$ в строке A$. Это значение копируется в пару регистров BC, и программа возвращается в БЕЙСИК.

* * *

Заканчивая печать книги Дж. Хардмана и Э. Хьюзона "40 лучших процедур", нам хотелось бы дать небольшой комментарий, который касается формата программных переменных в "Спектруме". Дело в том, что процедуры, представленные в этом последнем заключительном блоке широко оперируют с ними. Те, кто не имеют фирменную инструкцию по "Спектруму" (книга Виккерса), могут быть с этим форматом и не знакомы, а мы в своих работах до сих пор как-то к этому вопросу не обращались.

Те, кому этот вопрос интересен, могут прочитать комментарий на стр. 44.

Формат данных в "Спектруме"

Комментарий к стр. 43

Данные в "Спектруме" хранятся в виде переменных и массивов в специально выделенной для этого области памяти. Эта область начинается непосредственно за областью, в которой размешается текст БЕЙСИК-программы.

На начало области программных переменных указывает двухбайтная системная переменная VARS. Она расположена по адресу 23267 (5AE3H).

Конец области программных переменных задан специальным маркером - это байт, значение которого равно 80H (128),

"Спектрум" различает несколько разных типов переменных. Это:

- обычная числовая переменная, имя которой состоит из одной буквы, например x;
- числовая переменная, имя которой состоит из более, чем одной буквы, например row;
- числовой массив, например a(5) или b (3,3,40);
- переменные, управляющие циклами FOR. .. NEXT, например i;
- строковые переменные, например a$;
- строковые массивы, например b$(10,40);

Числовая переменная с именем из одной буквы.

Занимает 6 байтов. В первом байте хранится ее имя. В последующих пяти - ее значение в интегральной форме. Об интегральном представлений действительных чисел см. "Первые шаги в машинном коде". Первый байт имеет следующую раскладку:

0	1	1	б	у	к	в	а

На то, что это простая переменная указывает специфическое расположение первых трех битов.

Числовая переменная с именем более чем из одной буквы.

Ее первый байт имеет следующий формат:

1	0	1	б	у	к	в	а

Прочие байты имени (кроме последнего) имеют следующий формат:

0	б	у	к	в	а	.	.

Последний байт имени:

1	б	у	к	в	а	.	.

За именем следуют 5 байтов для выражения самого числа в интегральной форме.

Числовой массив.

Первый байт:

1	0	0	б	у	к	в	а

Байты 2, 3 содержат полную длину всех элементов (по 5 байтов на каждый элемент массива) плюс по 2 байта на каждую размерность массива плюс один байт на указание количества размерностей, т. е. здесь содержится указание на конец массива.

Байт 4 содержит размерность массива.

Байты 5,6 содержат количество элементов в первой измерении.

Если размерность массива более чем 1, то:

Байты 7, 8 содержат количество элементов во втором измерении;
и т. д.

После этого идут сами элементы массива по пять байтов на каждый элемент. Для многомерных массивов порядок следования данных следующий: b(1,1), b(1,2), b(1,3), b(2,1).....b(3,3)

Переменные цикла.

Первый байт:

1	1	1	б	у	к	в	а

Далее:
5 байтов - текущее значение;
5 байтов - конечное значение;
5 байтов - шаг;
2 байта - номер строки возврата;
1 байт - номер оператора в строке, к которому выполняется возврат.

Символьная переменная.

Первый байт:

0	1	0	б	у	к	в	а

Далее:
2 байта - длина строки.
x байтов - текст строки.

Символьный массив.

Первый байт:

1	1	0	б	у	к	в	а

Далее:
2 байта - указание на конец массива;
1 байт - размерность;
2 байта - длина в первом измерении:
2 байта - длина в последнем измерении;
Далее:
по одному байту на каждый элемент.

ЭЛЕМЕНТАРНАЯ ГРАФИКА В МАШИННЫХ КОДАХ

(Глава из книги "Элементарная графика")

Сегодня мы представляем Вам наше новое издание. Оно посвящено графике "Спектрума" и будет выпущено в четырех томах.

Первый том называется "Персональный компьютер ZX-СПЕКТРУМ. Элементарная графика".

Вашему вниманию предлагается одна глава из этой книги.

Книга готова к изданию и мы ждем заявок и предложении от организаций, способных приняться за ее издание и распространие на взаимовыгодных условиях.

ЭЛЕМЕНТАРНАЯ ГРАФИКА В МАШИННЫХ КОДАХ

Когда мы слышим сочетание слов "компьютерная графика", то нам сразу представляются сложные многоцветные трехмерные изображения и желательно, чтобы при этом что-то двигалось, лучше, если побольше, побыстрее и как можно более плавно.

Все это, конечно же так, но начинается компьютерная графика, тем не менее, не с этого. Когда Вы даете команду компьютеру PRINT "*" и он это делает, Вы уже работаете с графикой, хотя об этом и не задумывались. Можно считать так, что как только Вы делаете что-то, что приводит к изменению изображения на экране Вашего телевизора или монитора, Вы уже занимаетесь компьютерной графикой, особенно если Вам понятно, почему эти изменения происходят именно так, а не иначе и в какой-то степени можете ими управлять.

Итак, если Вы из БЕЙСИКа напечатаете звездочку на Вашем экране, то вам потребуется изрядная доля воображения для того, чтобы считать, что это компьютерная графика и убедить своих друзей, что у Вас есть дизайнерские способности. А что, если вы сделаете то же самое из машинного кода? А если при этом Вы ее не напечатаете, а нарисуете по точкам? Все дело принимает совсем другой оборот, не правда ли? Итак, все дело не в терминах, а в целенаправленности Ваших усилий, в способности задумать что-то и найти способы, как это реализовать.

Если вы не нашли до сих пор достаточно времени, чтобы освоить программирование в машинных кодах, то не только сузили круг своих технических возможностей, но и ограничили возможности для самовыражения и для дальнейших творческих поисков. Наш пример с печатанием звездочки здесь как раз и служит для того, чтобы дать представление о том, что и в графике значение имеет не только конечный результат, но и путь, который к нему привел.

В этой главе мы попробуем дать Вам те основы, которые необходимы дли того, чтобы начать эксперименты с графикой из машинного кода. Как и в любом другом вопросе, связанном с программированием на "Спектруме", мы не надеемся дать полную и исчерпывающую картину. Как и всегда, "ИНФОРКОМ" видит главную задачу в том, чтобы помочь сделать первый шаг, а дальше Вы сами раскроете свои таланты.

Первое, что нам потребуется - это вспомнить концепцию потоков и каналов "Спектрума". Те, кто более глубоко заинтересуются этой концепцией, могут найти информацию в "ZX-РЕВЮ" (N12, 1991г., с.227), мы же здесь рассмотрим этот вопрос в минимальном объеме хотя бы потому, что работая в БЕЙСИКе, Вы можете и не задумываться о потоках и каналах, а вот программируя в машинных кодах, без них не обойтись, коль скоро речь идет о графике, а значит о способах выдачи информации на экран.

Стандартными каналами "Спектрума" для вывода информации являются каналы:

- "K" - нижние две строки экрана (системное окно).
- "S" - главная часть экрана;
- "P" - стандартный ZX-принтер.

К этим каналам стандартно подключены потоки:
- поток #0 - к каналу "K";
- поток #1 - тоже подключен к каналу "K";
- поток #2 - подключен к каналу "S";
- поток #3 - к каналу "P"

Таким образом, оказываются идентичными следующие команды ввода/вывода:
PRINT #0 "Hello"; A$ - то же самое, что и INPUT "Hello": A$.
PRINT "Hello" - то же самое, что и PRINT #2 "Hello".
LPRINT "Hello" - то же самое, что и PRINT #3 "Hello".

Номер, стоящий после знака и в вышеприведенных примерах, является номером потока. Поскольку эти потоки подключены стандартно и переподключены быть не могут, мы программируем на БЕЙСИКе и используем операторы INPUT, PRINT, LPRINT без указания номера потока.

Это то, что касалось стандартных каналов и потоков, но они могут быть и нестандартными. Так, если Вы работаете в локальной сети, то сеть становится еще одним каналом, к которому вы подключите поток.

Вы знаете, что "Спектрум" может в любой момент времени выполнять только одно дело. Например, либо он печатает на экране, либо на принтере, одновременно выдавать информацию и туда и туда он не может, поэтому в любой момент времени задействован только один канал ввода/вывода и, соответственно, только один поток, связанный с ним. Этот канал и этот поток называются текущими. В большинстве случаев, когда Вы работаете с компьютером, текущим является канал "K", несколько реже канал "S".

Программируя в машинном коде, переключаться с канала "K" на "S" и наоборот - очень просто. О том, какой канал является текущим в данный момент, несет информацию нулевой бит системной переменной TVFLAG (5C3CH - 23612). Когда он выключен (равен 0), используется канал "S", а когда включен - "K".

Две небольшие процедуры продемонстрируют разницу в их использовании (листинг 1).

Не менее просто переключаться с каналов "S" или "K" на канал "P". Здесь тоже достаточно изменить один бит. Это первый бит системной переменной FLAGS (5C3BH - 23611). Он должен быть выключен для каналов "S" и "K", но включен для канала "P" (листинг 2).

Вы можете также изменить текущий канал, переключившись на другой поток. Это можно сделать вызовом процедуры ПЗУ, называющейся CHAN_OPEN и находящейся по адресу 1601H (5633). Перед тем, как ее вызывать, следует в аккумуляторе установить номер желаемого потока (листинг 3).

Нам необходимо знать эти азы потому, что если мы используем для печати из машинного кода команду процессора RST 10H, то должны иметь в виду, что она выдает информацию ТОЛЬКО В ТЕКУЩИЙ КАНАЛ. Прежде, чем Вы дадите компьютеру команду, что бы Вы хотели, чтобы он напечатал, надо сначала определиться, куда он будет это печатать и как.

Команду RST 10H Вы можете использовать для печати любых символов, будь то символ стандарта ASCII или графический символ. Это может быть управляющий символ и даже токен ключевого слова стандартного БЕЙСИКа. Поместите код того, что хотите напечатать, в аккумулятор и дайте команду RST 10H. При этом обычные символы займут одно знакоместо, управляющие коды сделают то, что им положено, а токены ключевых слов будут развернуты и займут столько знакомест, сколько букв в этом ключевом слове. Вам надо также знать, что команда RST 10H никогда не портит содержимое регистров процессора, кроме BC', DE' (альтернативные) и аккумулятора, который портит не всегда.

Печать чисел.

1. Целые числа от 0 до 9.

Выполнить печать целого числа от 0 до 9 можно двумя способами.

Во-первых, вы можете напечатать его обычным способом, как и любой другой символ. Для этого поместите в аккумулятор процессора его код (код 0 - 48 (30H).... код 9 - 57 (39H)) и дайте команду RST 10H.

Во-вторых, можно это число напечатать, загрузив в аккумулятор не его код, а само число, но в этом случае печать следует выполнять не командой RST 10H, а вызовом специально предназначенной процедуры ПЗУ - CALL 15EFH (десятеричный адрес - 5615), она называется OUT_ CODE. Это, конечно удобнее, но при своей работе эта процедура портит регистр E (имейте это в виду).

Возможен и промежуточный вариант, когда Вы засылаете в аккумулятор само число, а не его код, затем прибавляете к нему 30H (получаете его код) и затем даете RST 10H.

```
LD A,n
ADD A, 30H
RST 10
```

По расходу памяти это то же самое, что и CALL OUT_CODE, но не портит регистр E.

2. Целые числа от 0 до 9999.

Для печати целых чисел, меньших чем 10000, введите это число в регистровую пару BC и вызовите процедуру ПЗУ OUT_NUM_1. Она находится по адресу 1A1BH (6683).

Если Ваше число содержится в виде двух байтов в известном Вам адресе памяти, то Вы можете воспользоваться процедурой ПЗУ OUT_NUM_2 (1A28H = 6696). В этом случае перед вызовом процедуры надо в регистровую пару HL заслать адрес, в котором находится Ваше число.

И в том и в другом случае, если Вы попробуете через эти процедуры распечатать число, которое больше 9999, результат будет неверным.

Листинг 1.

Демо_S

```
213C5C          LD HL,TVFLAG
3600            LD (HL),00          ; Выключили бит 0 системной переменной TVFLAG.
3E2A      LOOP  LD A,42             ; Загрузили в аккумулятор число 42 (код символа "*")
D7              RST 10H             ; Выдали на печать по текущему каналу то,
                                    ; что находится в аккумуляторе.
18FB            JR LOOP             ; возврат для повтора.
```

Демо_K

```
213C5C          LD HL,TVFLAG
3601            LD (TVFLAG),01      ; Включили бит 0 системной переменной TVFLAG.
3E2A      LOOP  LD A,42
D7              RST 10H             : Печать символа "*".
18FB            JR LOOP             ; возврат для повтора.
```

Листинг 2.

Демо_P

```
213B5C          LD HL,FLAGS
CBCE            SET 1,(FLAGS)       ; Включили бит 1 системной переменной FLAGS.
0600            LD B, 00            ; Обнуление счетчика
                                    ; (подготовка к печати
                                    ; 256-ти символов).
3E2A      LOOP  LD A,42
D7              RST 10H             ; Печать символа "*".
10FB            DJNZ LOOP           ; Возврат для повтора,
                                    ; пока счетчик не достигнет нуля.
C9              RET                 ; Возврат в вызывающую программу
```

3. Целые числа от 0 до 65535.

В этом случае Ваше число тоже должно быть помещено в регистровую пару BC, но в

отличие от предыдущего случая необходимо делать не один вызов процедур ПЗУ, а два.

Сначала оно должно быть конвертировано в интегральную (пятибайтную) форму и помещено на стек калькулятора, это делается вызовом процедуры STACK_BC (2D2BH = 11563). И только после этого оно может быть напечатано, как действительное число с плавающей точкой. Это выполняется вызовом процедуры PRINT_FP, расположенной по адресу 2DE3H (11747).

4. Отрицательные целые числа.

Знак "минус" имеет код 2DH. Поместите его в аккумулятор, выполните RST 10H и абсолютную величину числа печатайте, как показано выше.

5. Произвольные действительные числа (числа с плавающей точкой).

Такое число занимает 5 байтов и хранить его ни в каком регистре, ни в регистровой паре невозможно. В этом случае Вами должны быть приняты меры для того, чтобы предварительно разместить его на вершине стека калькулятора. Когда это сделано, вызов процедуры PRINT_FP (2DE3H=11747) напечатает его на экране.

Здесь надо сделать пару предупреждений для тех, кто работает с калькулятором "Спектрума", зашитым в ПЗУ.

Во-первых, после работы процедура PRINT_FP, число со стека калькулятора снимается и, если Вам оно еще может потребоваться, то позаботьтесь предварительно продублировать вершину стека (соответствующая команда в сводке команд калькулятора имеется - см. "Первые шаги в машинном коде". М.: "ИНФОРКОМ", 1990г., т. 1).

Во-вторых, в процедурах ПЗУ, обслуживающих калькулятор, есть ошибка. Она заключается в том, что это число при вызове PRINT_FP снимается со стека не всегда. Если Ваше действительное число находится в диапазоне от -1 до +1 (ноль исключается), то вместо вашего числа на вершине стека остается 0. Обращайте на это внимание.

Печать символьных строк.

У Вас есть по крайней мере три способа печатать текстовые сообщения, если вы работаете в машинном коде. Но во всех случаях этот текст должен храниться в памяти компьютера и начинаться с известного Вам адреса.

Самый простой метод состоит в следующем. Регистровая пара DE должна содержать адрес, с которого начинается ваша символьная строка, а в регистровой паре BC необходимо предварительно установить длину этой строки. Печать выполняется вызовом процедуры PR_STRING, которая находится по адресу 203CH (8252).

Во-вторых, символьная строка может быть Вами получена в результате работы встроенного калькулятора. В этом случае она находится на вершине стека и может быть напечатана прямо оттуда вызовом процедуры ПЗУ PR_STR_1 (адрес 2036H-8246).

Третья возможность - самая мощная и именно она применяется в большинстве случаев в игровых, прикладных и системных программах.

У Вас может быть целый набор из N различных сообщений, хранящихся в виде таблицы. И Вы, допустим, хотите напечатать k-ое сообщение. Тогда можете действовать следующим образом:

- установите в аккумуляторе число k-1:
- установите в регистровой паре DE адрес, указывающий на байт, находящийся перед первым байтом самого первого сообщения из Вашей таблицы сообщений. Имейте в виду, что этот байт должен быть больше или равен 80H, т.е. старший (седьмой) бит в этом числе должен быть включен (он явится маркером начала текстового сообщения);
- вызовите процедуру PO_MSG (0C0AH-3082) и Ваше сообщение будет напечатано на экране.

Впрочем, у этого метода есть ряд ограничений и требований. Их необходимо также иметь в виду при программировании:

- недопустимо использование графических символов или токенов ключевых слов БЕЙСИКа, т.е. коды печатаемых символов должны быть менее 128 (80H). Впрочем, можно ведь и поменять символьный набор, заменив на время символы ASCII на нужные вам графические шаблоны;

- во-вторых, Вам надо при заполнении таблицы сообщений сделать так, чтобы последний символ каждого сообщения имел включенный 7-ой бит, тем самым он будет служить маркером конца 1-го сообщения и компьютер всегда по номеру, заданному в аккумуляторе, найдет нужное Вам сообщение;

- в таблице не может быть пустых строк.

На что надо обратить внимание!

Работая с графикой из машинного кода, Вам надо иметь в виду некоторые особенности, связанные с работой встроенного калькулятора. Дело в том, что он не вполне свободен от разнообразных ошибок и неточностей, а они могут доставить головную боль начинающему программисту. Совсем другое дело, когда он предупрежден.

Итак, во-первых, если вам необходимо будет выдавать на экран символы блочной графики (коды с 80H по 8FH) - эти символы расположены на цифровом ряду клавиатуры, - то коррумпируются (портятся) системные переменные, расположенные в адресах с 5C92H (23698) по 5C99H (23705).

Те, кто читал "Первые шаги в машинном коде", знают, что здесь хранятся две первые ячейки памяти встроенного калькулятора (из шести стандартных). Это ячейки M0 и M1. Почти во всех случаях вам эта порча безразлична, но если вы активно работаете с калькулятором и именно они Вам и нужны, то примите меры по сохранению их значений в другом месте, например на стеке или в других ячейках.

Во-вторых, при печати чисел мы довольно активно пользовались процедурой ПЗУ PRINT_FP, которая распечатывает содержимое вершины стека калькулятора, а она во время своей работы "портит" все шесть ячеек памяти калькулятора, т.е. все содержимое системной переменной MEMBOT с адреса 5C92H (23698) по адрес 5CAFH (23727). Опять же для Вас это почти всегда безразлично, за исключением тех редких случаев, когда вы оставили там на хранение данные без присмотра.

А вот третий случай довольно часто встречается у тех, кто много работает со встроенным калькулятором. Дело в том, что в таблице системных переменных есть переменная под названием MEM, она расположена в адресе 5C68H (23656) и занимает 2 байта. В ней хранится адрес, по которому расположена память калькулятора, т.е. адрес системной переменной MEMBOT. Если Вам достаточно тех шести ячеек памяти, которые есть стандартно, то нет проблем. А если же Вам их мало, то Вы создаете их побольше, для чего меняете адрес MEMBOT и, естественно, указание на нее, содержащееся в MEM. Это нормальная, часто встречающаяся операция. Так вот, после такой замены процедура PRINT_FP правильно работать не сможет.

Управляющие символы.

Набор символов "Спектрума" включает в себя стандартные символы ASCII - их коды с 20H (32) по 7FH (127). Символы блочной графики, графики пользователя и токены ключевых слов БЕЙСИКа - коды с 80H (128) по FFH (255). Это оставляет вне поля нашего зрения символы с 0 по 1FH (31). Что же расположено там?

А здесь расположены так называемые управляющие символы, их еще называют управляющими кодами (символами их называть и не хочется, ведь их нельзя напечатать и увидеть). Хоть сами они на экране и не воспроизводятся, зато с их помощью можно управлять процессом печати тех символов, которые могут быть напечатаны.

Мы начнем с управляющего кода 16H. Он называется AT_CONTROL и определяет координаты позиции печати так же, как и оператор AT в БЕЙСИКе.

Задействуются управляющие коды, как и любые другие символы, путем выставления этого кода в аккумуляторе процессора и подачей команды RST 10H. Это означает, что они могут быть как бы "напечатаны" вместе с прочими символами и могут быть включены в длинные символьные строки.

БЕЙСИКовский оператор AT требует после себя указания двух параметров - координат позиции печати по вертикали и горизонтали. Управляющий код AT требует то же самое. Оба параметра вводятся тем же способом - вводом параметра в аккумулятор и выдачей команды RST 10H.

Приведенная распечатка показывает машиннокодовый аналог оператора БЕЙСИКа PRINT AT 5,1; .

DEMO AT_5_4

```
3E16   LD A, 16H
D7     RST 10H     ; PRINT AT ...
3E05   LD A,05
D7     RST 10H     ; 5......
3E04   LD A,04
D7     RST 10H     ; 4.....
```

Не имеет никакого значения, сколько и каких машиннокодовых команд будет выдано между тремя вышеприведенными командами RST 10H. "Спектрум", если выдал управляющий код 10H, далее всегда будет помнить, что два ближайших числа, проходящих через RST 10H, являются параметрами управляющего кода AT_CONTROL.

Более часто этот управляющий код находит применение, когда речь идет не о печати в фиксированной позиции, а в позиции, заданной программными переменными. Ниже показан машиннокодовый аналог команды PRINT AT D, E (здесь D и E - содержимое соответствующих регистров процессора.)

DEMO AT_D_E

```
3E16   LD A,16H
D7     RST 10H     ; PRINT AT ...
7A     LD A,D
DT     RST 10H     ; D, ....
7B     LD A,E
D7     RST 10H     ; E. ....
```

Следующий управляющий код, который мы рассмотрим - это код 17H. Он называется TAB_CONTROL - код управления табуляцией. До некоторой степени он тоже аналогичен оператору БЕЙСИКа TAB и указывает на номер позиции печати в текущей строке.

За ним также следуют 2 параметра (это не ошибка - именно 2 параметра, хотя БЕЙСИКовский опыт учит, что для позиции печати достаточно и одного). Дело в том, что первый параметр - это младший байт координаты позиции печати, а второй параметр - ее старший байт. Поскольку экран "Спектрума" имеет всего лишь 32 символа по ширине, сразу и не понятно, зачем еще нужен какой-то старший байт? Но дело в том, что печать ведь может идти не только на экран. А если на принтер? Да, ZX-принтер тоже имеет только 32 символа в строке, но ведь возможна печать и на широкий матричный принтер, через интерфейс. Внимательный читатель может спросить, где мы видели принтер, печатающий в строку более 255 символов, но это уже вопрос риторический. Неважно, есть он или нет, но возможность работы с ним в компьютере предусмотрена. Более того, понятие "печать", как способ выдачи информации, может предусматривать не только стандартные каналы типа экрана или принтера. Возможны ведь и пользовательские каналы типа файлов на диске или в оперативной памяти, в которых запись может иметь и десятки тысяч символов в строке.

Но мы имеем дело все же с экраном, поскольку речь идет о графике, и для печати на экране то, что находится в старшей байте, не имеет никакого значения.

Дело в том, что когда Вы задаете параметр TAB, компьютер берет из него остаток от деления на 32, а старший байт кратен 256 и, естественно, кратен 32, поэтому его содержимое влияние при печати на экран и не оказывает.

Нижеприведенная процедура показывает управляющий код TAB_CONTROL в действии. Первый параметр предполагается переменным и берется из регистра E. Выдавая второй параметр, мы обнулили его, использовав для этого обнуление аккумулятора через XOR A, но раз этот байт роли не играет, можно было XOR A и не давать, а отправить в этот параметр то, что было в аккумуляторе к этому моменту, неважно, что это было.

```
3E17   LD A,17
D7     RST 10H     ; PRINT TAB...
7B     LD A,E
D7     RST 10H     ; E....
AF     XOR A
D7     RST 10H     ; 0 ...
```

Код CHR 6 называется COMMA_CONTROL и выполняет то же, что и запятая оператора PRINT. Код подается точно так же, как и прочие, введением числа 6 в аккумулятор и выдачей его через RST 10H. В результате его действия в качестве позиции печати устанавливается 0 или 15, в зависимости от того, в какой половине экрана печаталось последнее знакоместо.

Код 0DH (13) - аналогичен ENTER. Выдается так же и обеспечивает прекращение печати в текущей строке и переход в начало следующей.

Код 08 - называется BACKSPACE_ CONTROL, он выполняет смещение позиции печати на одно знакоместо влево.

Команда процессора RST 10H -довольно гибкая и имеет самое разнообразное действие. С ее помощью можно не только управлять позицией печати символов на экране, но и управлять цветами того, что Вы воспроизводите, т.е. выдавать коды цветовых атрибутов. Выполняется это точно также путем предварительной установки в аккумуляторе управлявшего кода и передачей его, а затем передачей параметра. Мы не будем подробно останавливаться на цветовых атрибутах и привели их в Таблице 1.

Таблица 1.

Управляющий код	Параметры	Комментарии
06 COMMA_CONWR 08 BACKSPACE 0D ENTER	- - -	
10 INK_CONTROL 11 PAPER_CONTR	00-черный 01 -синий 02 -красный 03-пурпурн. 04- зеленый 05- голубой 06 -желтый 07-белый 08 -транспарантный 09 -контрастный	Параметр 08 (транспарантный[1]) означает, что цвет печати не устанавливается, а берется тот, который уже есть в данной позиции печати. Параметр 09 (контрастный) устанавливает цвет INK или PAPER контрастным к уже существующему в данной позиции противоположному цвету.
12 FLASH_CONTROL 13 BRIGHT_CONTR	00 - выключен 01 -включен 08 - транспарантный	См. выше.
14 INVERSE_CONTR 15 OVER_CONTROL	00- выключен 01-включен	
16 AT_CONTROL 17 TAB_CONTROL	Строка, столбец. Младший байт, старший байт.	При выдаче на экран старший байт может быть любым. Можете, например, использовать эту ячейку памяти для хранения какой-либо однобайтной переменной.

Пример применения кодов управления цветовыми атрибутами показывает, как выполняется печать красной звездочки на желтом фоне. Обратите внимание на то, что установленные таким способом цветовые атрибуты остаются действующими до того, как

[1] Transparent - прозрачный (Прим. OCR)

будут изменены или до окончания действия оператора БЕЙСИКа, из которого вызывалась данная процедура в машинных кодах.

```
3E10   LD A,10H
D7     RST 10H      ; PRINT INK...
3E02   LD A, 02
D7     RST 10H      ; 2; ....
5E11   LD A,11H
D7     RST 10H      ; PAPER...
3E06   LD A, 06
D7     RST 10H      ; 6;....
3E2A   LD A, 2A
D7     RST 10H      ; "*";
```

Другие приемы управления позицией и цветом печати.

В "Спектруме" есть возможность задействовать некоторые процедуры ПЗУ и управлять системными переменными для того, чтобы достигать тек же эффектов, которые дает применение управляющих кодов. Рассмотрим эти приемы.

PRINT AT B,C (где B и C -содержимое одноименных регистров процессора) может быть выполнено вызовом процедуры AT_B_C, находящейся по адресу 0A9BH (2715).

PRINT TAB A (где A - содержимое аккумулятора) выполнимо вызовом процедуры TAB_A (0AC3H=2755).

Управляющий код 06 (COMMA_CONTROL) эмулируется вызовом PO_COMMA (0A5FH=2655).

Эти три приема действуют не только на канал "S", но и на канал "K", т.е. с их помощью можно печатать информацию и в INPUT-строке.

У Вас есть также достаточно простой способ управлять цветовыми атрибутами INK, PAPER, BRIGHT, FLASH. Все, что для этого требуется - внести изменения в системную переменную ATTR_T (5C8FH = 23695), отвечающую за статус временных атрибутов экрана, эта системная переменная имеет следующую раскладку:

Биты 0...2 - цвет INK (от 0 до 7)

Биты 3...5 - PAPER (от 0 до 7)

Бит 6 - статус BRIGHT (0 или 1)

Бит 7 - статус FLASH (0 или 1)

Все, что требуется для установки нужных комбинации цвета - это заслать в эту системную переменную число, определяемое по формуле:

128*F + 64*B + 8*P + 1, где:

- F - статус FLASH:
- B - статус BRIGHT;
- P - номер цвета PAPER;
- I - номер цвета INK.

Если же Вы хотите, чтобы какие-то цветовые атрибута были транспарантными, системной переменной ATTR_T Вам уже недостаточно и надо воспользоваться системной переменной MASK_T (5C90H = 23696). Ее раскладка точно та же, что и у ATTR_T. Биты, соответствующие атрибуту, который Вы хотите сделать транспарантным, надо включить.

Итак, рассмотрев, каким образом из машинного кода выполняется печать графики низкого разрешения, мы переходим к графике высокого разрешения, но для этого надо сначала хорошо разобраться со структурой экрана и, главное, понять, как раскладка экрана связана с организацией экранной памяти. Отсюда мы сделаем шаг к наиболее рациональным приемам по изменению экранных форм из машинного кода.

Организация экранной памяти.

Область оперативной памяти "Спектрума", в которой находится графическое изображение, видимое на экране в качестве включенных и выключенных точек (пикселов), называется экранной областью памяти.

Эта область занимает адреса с 4000H по 5AFFH (16384 - 23295). Причем, она состоит

из двух областей. В первой с 4000H по 57FFH (с 16364 по 22537) расположено растровое изображение, состоящее из черных и белых неокрашенных точек (дисплейный файл). Во второй, с 5800H по 5AFFH (с 22528 по 23295) - цветовая информация (файл атрибутов).

Размер всей экранной области - 6912 байтов. Из них 6144 байта - дисплейный файл (черно-белый) и 768 байтов - файл атрибутов (цвет).

Как получены эти числа 6144 и 728?

Вы знаете, что полный экран "Спектрума" может иметь 24 ряда по 32 символа в ряду, т. е. всего 24*32=728 знакомест. Каждое знакоместо образовано из 64-х пикселов (8 линий по 8 точек). На каждую линию достаточно одного байта (8 битов соответствуют 8 точкам) и, следовательно, для растровой графики одного знакоместа необходимы 8 байтов. Так, на 728 знакомест необходимо 728*8=6144 байта для хранения черно-белой информации.

Информация о цвете хранится более экономно. Каждому знакоместу отдан один байт, определяющий окраску всех 64 его пикселов. Три младших бита определяют цвет INK этого знакоместа, еще три бита определяют цвет PAPER и по одному биту отдано признакам BRIGHT и FLASH. Отсюда вытекает одно из самых неприятных ограничений графики "Спектрума" - в пределах одного знакоместа невозможно иметь одновременно более 2-х цветов.

А теперь давайте немного поэкспериментируем. Очистите экран - CLS. Окрасьте бордюр в голубой цвет - BORDER 5. Это нужно, чтобы точно видеть результат следующих манипуляций и дайте команды, задающие самый первый байт экранной области:

```
POKE 16384,255
POKE 16384,15
POKE 16384,85
```

После каждой команды вы увидите изменения в левом верхней углу экрана, а точнее - в первой линии первого знакоместа. На рис. 1 показана эта линия укрупненно и Вам должно быть понятно, что и почему там происходит. Вы видите, что раскладка пикселов в этой линии соответствует раскладке битов в байте 16384, отвечающем за эту линию.

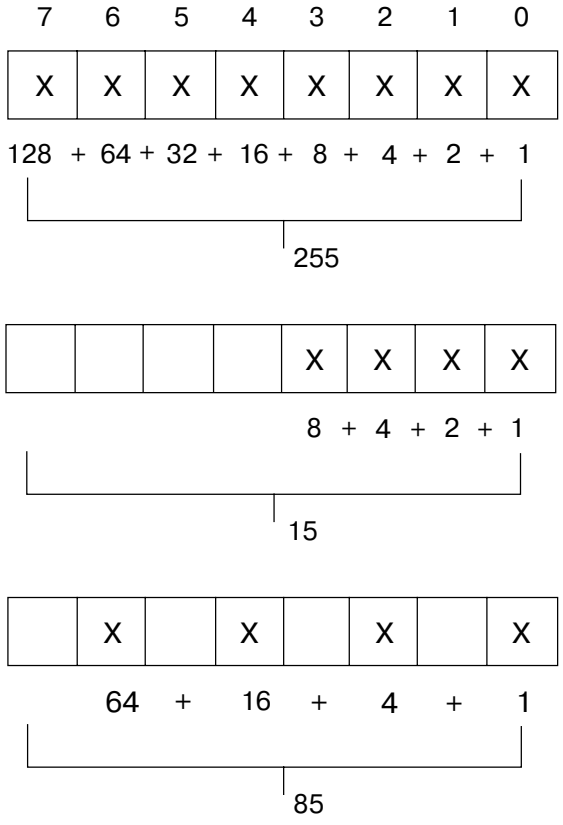


Рис. 1.

Теперь попробуем закрасить вторую линию в первом знакоместе. Логично предположить, что POKE в следующую ячейку памяти сделает это. Дайте POKE 16385,85. Получилось совсем не то - включилась первая линия второго знакоместа, далее третьего и так далее, до 32-го. Может быть, попробуем их пропустить и дадим POKE сразу в 33-ий адрес:

```
POKE (16384*32), 85
```

И опять ничего не получилось. Вместо второй линии в первом знакоместо включается первая линия в первом знакоместе, но во втором ряду. И так будет продолжаться, пока Вы не пройдете все 32 знакоместа в 8 рядах экрана, т.е. 32*8=256 адресов. Только дав POKE (16384 + 256), Вы получите то, что хотели. На первый взгляд, такое соответствие между точками экрана и ячейками экранной памяти выглядит достаточно нелепым и нелогичным. Но это не совсем так. Как окажется чуть ниже, это не только не усложняет жизнь, а при работе из машинного кода даже упрощает - надо только хорошо все понять.

Теперь, покопавшись в памяти, Вы наверняка вспомните, что уже много раз видели при загрузке заставок игровых программ, как экран прорисовывается постепенно, строчка за строчкой. В этот момент Вы и видели соответствие между структурой экрана и экранной памятью. Хотите повторить, пожалуйста:

```
10 CLS: BORDER 5
20 FOR i=16384 TO (16384+256*8)
30 POKE i, 255
40 NEXT i
```

Чтобы не утомлять Вас длительным ожиданием, мы сделали это только для 8 первых рядов экрана.

Научившись изображать на экране точки и тире без помощи операторов PRINT и PLOT, мы уже сделали большое дело и развязали себе руки, т.к. мы теперь умеем манипулируя с ячейками памяти делать графику, а машинный код именно и хорош, когда дело доходит до манипуляций с большими объемами памяти.

Но как насчет того, чтобы получить нормальный графический образ? Нет проблем, этот образ сначала надо где-то в памяти создать. Давайте воспользуемся готовым. Вы, конечно знаете, что в ПЗУ компьютера где-то имеется набор символов, в котором хранятся графические образы (шаблоны) всех букв и прочих знаков, вот мы возьмем оттуда образ буквы "A" и нарисуем ее на экране без PLOT или PRINT.

В адресах 23606, 23607 (5C36H) хранится 2-х байтная системная переменная CHARS, которая указывает на 256 байтов ниже, чем адрес, с которого начинается набор символов. Во-первых, давайте разберемся, почему она указывает ниже на 256 байтов, а не туда, куда надо. Все очень просто. Мы уже говорили о том, что первые 32 символа вовсе и не символы, а управляющие коды и им графические образы не нужны, все равно они не печатаются. А т.к. символы занимают по 8 байтов каждый, то на пропуск этих 32 управляющих кодов и ушло это снижение на 256 байтов. Зато теперь мы можем искать образ буквы A по ее номеру (по ее коду ASCII, который, кстати, равен 65(41H)).

```
10 LET base=PEEK 23606 + 256*PEEK 23607
20 LET addr = base+8+65
25 LET screen= 16384
30 FOR i=0 TO 7
40 LET pic=PEEK (addr+i)
50 POKE (screen+256*i), pic
60 NEXT i
```

Мы с Вами только что нарисовали букву "A", причем именно нарисовали, а не напечатали, да к тому же работали при этом только с переброской данных из одних участков памяти в другие. Вы обратили внимание на то, что засылали мы графику в дисплейный файл в строке 50 с шагом через 256 адресов? Давайте теперь рассмотрим, как то же самое будет выглядеть в машинном коде и почему так сложно, на первый взгляд, организован дисплейный файл.

Адреса экранной памяти обычно при работе с экраном хранятся в регистровой паре HL. При этом в H хранится старший байт адреса (High - старший), а в L - младший (Low), для того, чтобы увеличить адрес на 256 и перейти к нижележащей линии в том же знакоместе, оказывается достаточно увеличить на единицу старший байт. С этой задачей изящно справляется простая команда АССЕМБЛЕРа INC H. Увеличение же на единицу младшего байта адреса (INC L) переместит Вас на соседнее знакоместо вправо в той же линии. Видите, как все просто, и преобразование вышеприведенной БЕЙСИК-программы в

машинный код выглядит (листинг 4):

ЛИСТИНГ 4.

```
        LD DE,(23606)           ; Загрузили в DE содержимое системной переменной CHARS.
        LD HL,0041H             ; Загрузили в HL код буквы A.
        ADD HL,HL               ; Умножили его на 2.
        ADD HL,HL               ; Умножили его на 4.
        ADD HL,HL               ; Умножили его на 8.
        ADD HL,DE               ; Теперь HL указывает на начало шаблона буквы A.
        EX DE,HL                ; Освободили HL для работы с экраном.
        LD HL,4000H             ; Загрузили в HL адрес начала экранного файла.
        LD B,08                 ; Организуем счетчик на 8 шагов
LOOP    LD A,(DE)               ; Переброска содержимого из
        LD (HL),A               ; DE в экранный файл через аккумулятор.
        INC H                   ; переход к новой линии экрана.
        INC DE                  ; Переход к новой линии шаблона.
        DJNZ LOOP               ; Окончание цикла из 8-ми шагов.
        RET                     ; возврат.
```

Итак, все вроде бы просто и понятно, но Вы, конечно, обратили внимание на то, что все, что мы до сих пор делали, относится только к восьми первым символьным рядам экрана, а что же дальше? Ведь их всего 24.

Здесь тоже все просто, если представить себе, что экран состоит из трех несвязанных между собой областей, каждая из которых имеет по 8 рядов. Назовем эти трети экрана сегментами. Итак, экран состоит из трех сегментов, каждый из которых состоит из восьми рядов, каждый из которых состоит из тридцати двух знакомест, каждое из которых состоит из восьми линий, каждая из которых состоит из восьми пикселов.

Первый сегмент -16384 - 18431 (4000H - 47FFH)

Второй сегмент -18432 - 20479 (4800H - 4FFFH)

Третий сегмент - 20480 - 22527 (5000H - 57FFH)

Каждый сегмент занимает по 8*32*8=2048 байтов.

Если теперь развернуть регистровую пару HL в виде шестнадцати битов, то получится любопытная побитовая карта для дисплейного файла (см. рис. 2).

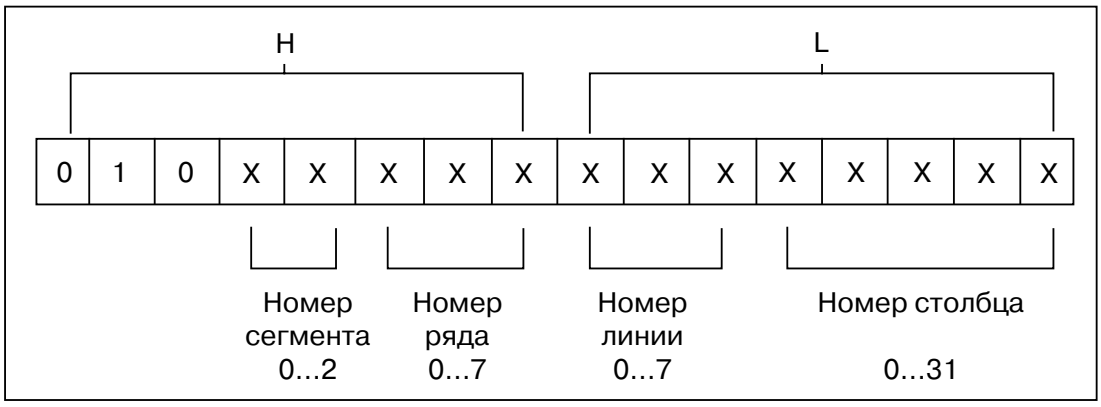


Рис.2

Однажды, в 1985 году сэра К. Синклера спросили, чем он объясняет столь невероятный успех своих компьютеров по сравнению с главными конкурентами "Коммодором", "Атари", "Амстрадом" и "Би-Би-Си Микро", если известно, что графика у них лучше, музыка богаче, надежность выше и дополнительных внешних портов больше? На это он ответил: "У меня гораздо проще доступ к памяти". И этим сказано все. В частности, организация экранной памяти в "Спектруме" была не последним фактором, обеспечившим ему поддержку со стороны сотен фирм, выпускающих программное обеспечение.

Давайте рассмотрим эти преимущества.

1. В отличие от многих других разработчиков К. Синклер нашел удачное решение, объединив в едином экране и графику высокого разрешения и символьную графику.

Нет необходимости переключаться на другой экран всякий раз, как надо построить диаграмму или написать текст. Это большой подарок программистам.

2. Решение о разделении черно-белой информации высокого разрешения и цветовой информации низкого разрешения в разные файлы позволило ему в 4-5 раз уменьшить объем экранной памяти (по сравнению с некоторыми конкурентами) и выделить много места для пользователя. Практически здесь имитируется цветная графика высокого разрешения, хотя она таковой строго говоря и не является - это был гениальный ход, высоко оцененный специалистами.

3. Эти решения позволили в значительной степени сократить размеры программ ПЗУ. Действительно, по отношению возможностей ПЗУ к его размерам вряд ли найдется машина, равная "Спектруму", хотя и у этого ПЗУ, как показал дальнейший опыт, еще были огромные резервы.

Теперь посмотрим, как на практике ухватились программисты например за такой простой элемент, как сегментирование экрана на три зоны при том, что экран является одновременно и графическим и символьным.

Во-первых, множество первоклассных программ используют для своей графики только один сегмент, оставив прочее на диалог с пользователем. Ведь стоит только вдуматься, что в таких программах, как TIR-NA-NOG, DUN DARACH, MARSPORT фирмы "GARGOYLE GAMES" огромные события происходят на участке памяти размером всего лишь 2K.

Во-вторых, стал традиционным прием, при котором графика занимает вроде бы весь экран и непрофессионал даже и не замечает, что два сегмента из трех заняты красочной статической графикой, в то время как все действие разворачивается лишь на одном сегменте (может быть и менее красочном). Малый размер памяти этого сегмента позволяет достичь высокого быстродействия, плавности переметения объектов, прекрасной реакции на действия пользователя, а общий эстетический эффект непроизвольно распространяется на весь экран. Вроде бы имитация, но какая!

В третьих, очень часто экранную память неиспользуемых сегментов начинают использовать для разных вспомогательных действий, для размещения временных таблиц, буферов и т.п., втискивая в небольшие размеры оперативной памяти огромное количество информации. Вы сами, возможно видели во время работы копировщика COPY-86M, как временно ненужная информация выбрасывается на экран в виде точек и тире в двух нижних сегментах экрана. Это же происходит и при работе кнопки MAGIC BUTTON в операционной системе TR DOS. Тот же прием применяют и в игровых программах, но там Вы этого не увидите, т. к. программист заблаговременно установил в цветовых атрибутах, относящихся к этим сегментам экрана, черный цвет INK и черный цвет PAPER. За черным цветом прячут иногда даже и машинный стек процессора (например в программе NETHEREARTH), что не только экономит оперативную память, но является еще и эффективным приемом защиты программы от внешнего вторжения, и многое-многое другое.

Файл атрибутов.

Теперь, рассмотрев работу с дисплейным файлом, перейдем к файлу атрибутов и посмотрим работу следующей программы:

```
10 PAPER 6; INK 0: BORDER 6: CLS
20 FOR i=1 TO 22
30 PRINT "ZX-SPECTRUM"
40 NEXT i
50 FOR i=22528 TO 23295
60 POKE i,15
70 NEXT i
```

Посмотрите, что произойдет, когда Вы запустите эту программу. Во-первых, на экране будет напечатан текст (черным цветом по желтому фону). Во-вторых, знакоместо за знакоместом, начнется изменение его цвета (белые буквы по синему фону). Чтобы понять, почему так происходит, нам надо изучить область памяти от адреса 22528 до адреса 23295 (5800H -5AFFH), которая и называется файлом атрибутов. Содержимое ячеек памяти в этой области определяет то, каким цветом будет окрашено то или иное знакоместо.

Структура этого файла очень проста. Каждому знакоместу экрана соответствует один байт памяти, а значение этого байта и определяет цвета этого знакоместа. Это означает, что когда вы засылаете (POKE) какое-либо число в ячейку памяти этой области, Вы изменяете цвет одного из знакомест экрана. Каждый байт из файла атрибутов хранит информацию о параметрах INK, PAPER, BRIGHT и FLASH.

Биты 0…2 - цвет INK (от 0 до 7)

Биты 3… 5 - PAPER (от 0 до 7)

Бит 6 - статус BRIGHT (0 или 1)

Бит 7 - статус FLASH (0 или 1)

Схематически это выглядит так:

7	6	5	4	3	2	1	0
F	B	P	P	P	I	I	I

Рис. 3

Для программирующего в машинном коде, конечно, чрезвычайно важно знать точно, какому знакоместу экрана соответствует какой байт файла атрибутов. Зная, что в каждом ряду 32 знакоместа, этот адрес вычислить нетрудно:

22528 + 32 * Y + X

Здесь Y и X - координаты знакоместа. Y - номер ряда сверху вниз, X - номер столбца слева направо. Как видите, в отличие от дисплейного файла, файл атрибутов организован вполне благоразумно: слева направо и сверху вниз.

Изменение цвета бордюра.

При работе в машинном коде здесь есть небольшая сложность, заключающаяся в том, что для изменения цвета бордюра требуется два действия, т.к. к этой цели ведут два пути. Дело в том, что тот существует текущий цвет бордюра (тот, который Вы в данный момент видите на экране) и есть установленный цвет бордюра (тот, который записан в памяти компьютера и будет установлен, как только ПЗУ компьютера перехватит управление от Вашей программы в машинных кодах, например при нажатии какой либо клавиши).

Ваша задача состоит в изменении обоих цветов и текущего и установленного, т. к. изменение только текущего будет иметь кратковременный характер, а изменение только установленного вообще никак не отразится на экране в данный момент.

Текущий цвет бордюра изменяется выдачей байта по 254-ому внешнему порту (бордюрная часть экрана является для "Спектрума" внешним устройством, подключенным к порту FEH). Это можно сделать даже из БЕЙСИКа, воспользовавшись командой OUT:

```
OUT 254, цвет
```

В машинном коде эта команда выглядит удивительно похоже:

```
LD A, цвет
OUT (FEH),A
```

Установленный цвет бордюра, с которым работает ПЗУ, постоянно хранится в отведенной для него ячейке памяти в области системных переменных. Ее название BORDER. Ее адрес 23624 (5C48H). Три бита этой переменной отвечают за цвет бордюра:

7	6	5	4	3	2	1	0
X	X	BRD	BRD	BRD	X	X	X

Рис. 4

Прочие биты занимаются другими делами, в частности, они задают цвет нижней части экрана, используемой для работы INPUT и пр.

Оба переключения одновременно можно сделать вызовом процедуры ПЗУ BORDER_A, расположенной по адресу 2297H (8855). Предварительно код цвета надо установить в аккумуляторе процессора. У нее есть и побочное воздействие. Если при вызове этой процедуры что-то содержится в системных нижних двух строках компьютера, то она обойдется с ними достаточно "гуманно". Чтобы бордюр не скрыл информацию, цвет INK этих строк будет установлен контрастным к цвету бордюра.

Владельцам 128-х машин.

"Спектрумы" со 128 килобайтной памятью имеют не одну, а две экранных области. (См. "ZX-РЕВЮ", N1, с.4; N2, с.26; М.: ИНФОРКОМ, 1991), каждая из которых может быть использована для хранения изображения, если Вы работаете в режиме 128K.

Первая экранная область, как обычно занимает адреса 4000H-5AFFH, а вторая расположена в адресах C000H - DAFF на седьмой странице оперативной памяти. Первая область называется нулевым экраном, а вторая - первым экраном. Очевидно, что только одна область из двух может быть воспроизведена на экране телевизора в данный момент.

Раскладка экрана-1 точно такая же, как и экрана-0 и мы можем считать, что разница состоит во-первых, в 15-ом байте (рис. 5), а во-вторых в том, что он расположен не на нулевой странице ОЗУ, а на седьмой.

Тот экран, который в настоящее время задействован, называется активным. Обратите внимание на интересную особенность первого (дополнительного) экрана. После того, как он был задействован, уже нет необходимости седьмой странице ОЗУ оставаться "впечатанной". Т.е. экран-1 может оставаться активным независимо от того, какие страницы ОЗУ "впечатаны" в данный момент.

Есть еще одна особенность: процедуры ПЗУ могут работать только с нулевым (основным) экраном. Они не умеют печатать и рисовать на втором экране. Ни PLOT, ни PRINT, ни DRAW, ни даже машиннокодовая команда RST 10H не работают с первым экраном. С ним не работают ни автоматический листинг, ни система редактирования, ни INPUT.

"Спектрум-128", как это ни странно, не содержит в своем ПЗУ-128 совершенно ничего, чтобы переключаться с одного экрана на другой и пользователю приходится самому писать для этого процедуру, например такую, как показано ниже, точками входа в нее являются метки SCR_0 и SCR_1, соответствующие тому, какой экран Вы хотите вызвать (листинг 5).

Примечание: Обязательный порядок всех переключений в 128-х машинах - сначала перенастраиваем BANK_M и только потом выдаем команду по порту 7FFD. Системные прерывания компьютера вызывают запуск его процедур ПЗУ, которые, заканчивая работу, выдают по порту 7FFD содержимое BANK_M. Так что, если Вы будете делать наоборот, есть высокая степень вероятности, что у Вас ничего не получится, правда, из машинного кода прерывания можно и отключать (см. "Первые шаги в машинном коде").

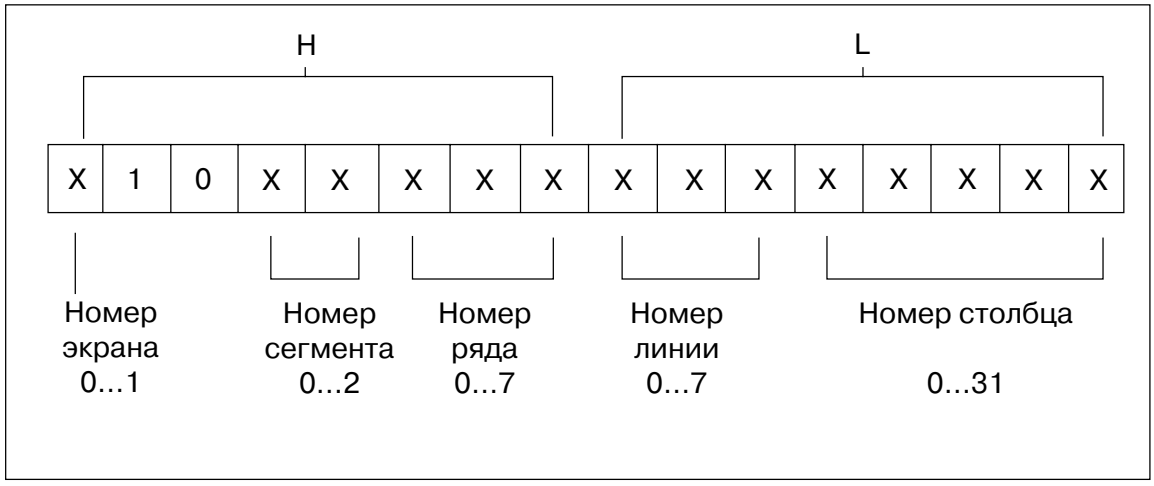


Рис. 5

ЛИСТИНГ 5.

```
1E00        SCR_0       LD E,00             ; Все биты выключены.
1802                    JR SELECT           ; Переход на переключение экрана.
1E08        SCR_1       LD E,08             ; Включен бит-3.
3A5C5B                  LD A,(BANK_M)       ; Системная переменная
                                            ; BANK_M (5B5CH) - служит
                                            ; в 128-килобайтных машинах
```

```
                                ; как указатель страниц
                                ; ОЗУ, ПЗУ, экрана и
                                ; режима (см. рис. 6).
                                ; За номер экрана отвечает
                                ; бит 3.
E6F7            AND F7          ; включили все биты в
                                ; аккумуляторе, кроме 3-го.
B3              OR E            ; третий бит включается
                                ; или выключается в зависимости
                                ; от того, через какую
                                ; точку мы вошли в
                                ; эту процедуру.
325C5B          LD (BANK_M),A   ; Переключили BANK_M на
                                ; экран-1. Но он еще не
                                ; активен. Это только подготовка.
01FDF7          LD BC,7FFD      ; Установили в BC номер
                                ; внешнего порта, служащего для
                                ; физического переключения страниц и режимов.
ED79            OUT (C),A       ; Переключение экрана.
C9              RET             ; Выход
```

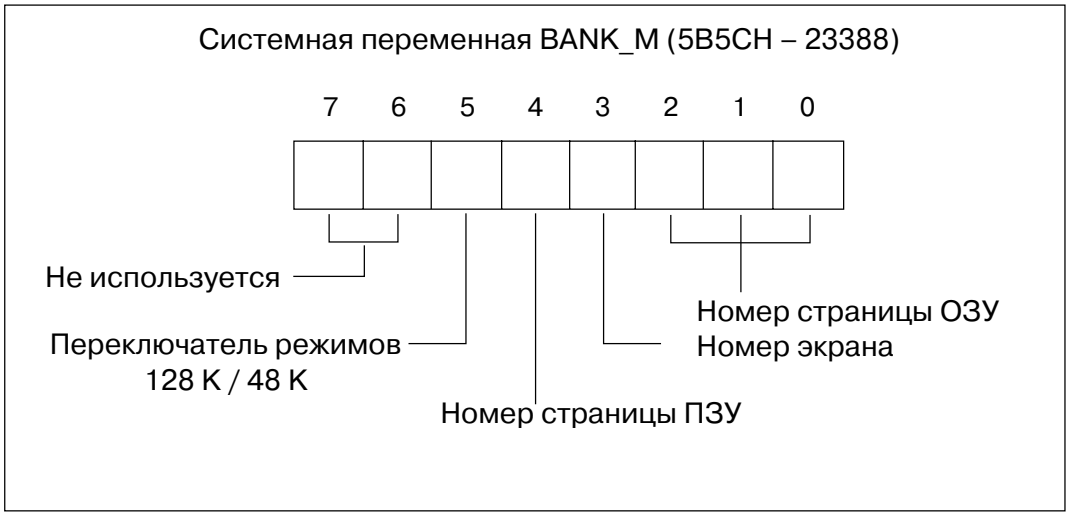


Рис.6

И еще одно предостережение для тех, кто работает с RAM-диском на 128-килобайтных машинах (о RAM-диске см. "ZX-РЕВЮ", N12,С.235-240; М.:ИНФОРКОМ, 1991.). Когда вы выгружаете что-либо в RAM-диск, используя для этого новую команду SAVE!, файлы сохраняются в памяти в виде стека - один над другим. Начало области - 1C000 и далее вверх до 1FFFF, потом с 3C000 по 3FFFF, с 4C000 по 4FFFF, с 6C000 по 6FFFF и, наконец, с 7C000 и вверх.

Одновременно с этим образуется второй стек. Он начинается в адресе 7EBFF и развивается вниз, В нем содержатся только имена и адреса файлов, выгруженных на RAM-диск. Короче говоря, этот стек содержит ту информацию, которую Вы получаете, воспользовавшись новой командой CAT! . Таким образом, эти два стека развиваются навстречу друг другу, встречаться они не должны. При появлении такой опасности компьютер выдаст сообщение об ошибке "4 Out of memory", стек каталога не может также опуститься ниже 7C000 - именно поэтому установлено ограничение на количество сохраняемых файлов - 562.

Итак, все сделано, чтобы эти стеки не встретились и не испортили друг друга, но ничего не сделано для того, чтобы они не затерли экран-1. Этот экран может быть испорчен как стеком файлов, так и каталожным стеком. И даже наоборот, работая с экраном-1, Вы можете испортить эти стеки. Будьте внимательны. Работая в машинном коде, использовать экран-1 можно только если Вы не очень активно используете виртуальный диск. Смело можете хранить до 216 файлов с суммарной длиной не более 64K.

Эмуляция команд БЕЙСИКа из машинного кода.

Здесь мы рассмотрим, как выполнить БЕЙСИК-овские команды, связанные с графикой, из машинного кода. Как правило, это несложная задача. Поскольку в ПЗУ имеются процедуры, способные это сделать, надо только знать, каким образом к ним следует обращаться.

CLS

Сначала займемся очисткой экрана. В машинном коде это можно сделать вызовом процедуры CLS, расположенной в ПЗУ по адресу 0D6BH (3435). В результате ее работы выключаются все пикселы на экране, а цветовые атрибуты устанавливаются такими, как установлено в системной переменной ATTR_P 5C8DH (23693). Ее побитовая раскладка та же, что и у системной переменной ATTR_T (см. выше). Данная процедура работает точно так же, как и соответствующая команда БЕЙСИКа.

Вы можете также очистить только нижнюю часть экрана (обычно это две нижние строки). Это делается вызовом процедуры CLS_LOWER, которая находится по адресу 0D6EH (3438). В отличие от основного экрана, эта область очищается с цветом бордюра, а не с цветом, установленным в ATTR_P.

Среди процедур ПЗУ есть еще одна, обеспечивающая более мощные возможности. Она называется CL_LINE (0E44H = 3652) и очищает заданное количество строк в нижней части экрана.

Перед ее вызовом в регистре B должно быть установлено количество строк, подлежащих очистке от 01 до 18H (от 1 до 24). Так, например, чтобы очистить 10 нижних строк, нужно предварительно в регистр B заслать число 0AH.

Эту процедуру можно применять двумя способами. Если нулевой бит системной переменной TVFLAG (5C3C = 23612) выключен, то при очистке используются цвета экрана, а если он включен, то используется цвет бордюра. Таким образом, здесь есть возможность очистить одновременно весь экран, включая и нижние две строки, в цвет ATTR_P. Процедура CLS сделать такого не может. В то же время, следует обратить внимание на то, что процедура CLS после очистки экрана инициализирует курсор и текущая позиция печати устанавливается в исходное положение - левый верхний угол. Процедура же CL_LINE этого не делает.

Скроллинг экрана.

Когда Вы пытаетесь напечатать что-либо за последней возможной позицией печати, то вместо результата получаете сообщение "scroll?" в нижней части экрана и компьютер ждет от Вас нажатие клавиши. Если это клавиша "N" или "BREAK", печать на экране прекращается с сообщением об остановке, в противном случае печать продолжается. Экран вроде бы заполнен, но печать возможна, происходит скроллинг вверх. Этот скроллинг происходит автоматически и длится до тех пор, пока вся информация, присутствовавшая на экране, когда выходило сообщение "scroll?", не скроется из виду за верхней кромкой экрана.

Такой вид скроллинга называется автоматическим и он существует на "Спектруме" как в БЕЙСИКе, так и в машинном коде.

Но возможен также и "ручной" скроллинг. Вызов процедуры CL_SC_ALL (0DFEH = 3582) "прокрутит" весь экран, но при этом возникнут несколько неожиданные эффекты:

- во-первых, позиция печати не изменится и останется там же, где была. С этим Вам придется разбираться вручную.

- во-вторых, если первая строка нижней части экрана (системного окна) не пуста или имеет цвет, отличный от цвета основного экрана, то результат может быть не совсем тем, какой Вы ожидаете.

Другой способ исполнения "ручного" скроллинга еще интереснее. Он базируется на использовании процедуры CL_SCROLL (0E00H 3584). С ее помощью можно "прокручивать" только часть экрана. Предварительно в регистр B надо занести количество строк, подлежащих скроллингу (обратите внимание на то, что изменено будет на одну строку

больше). Минимальное количество строк - 1. В этом случае нижняя строка экрана будет поднята на одно знакоместо вверх, а снизу появится чистая строка. Для того, чтобы переработать весь экран, следует установить число 17H. Таким образом, действие этой команды состоит в перемещении на одну позицию вверх "В" нижних строк и в очистке нижней строки.

PAUSE

БЕЙСИК-оператор PAUSE можно легко эмулировать в машинном коде. Единицей измерения времени в "Спектруме" являются пятидесятые доли секунды (потому, что частота переменного тока в нашей сети 50Гц. Если бы Вы жили в Англии, то у Вас в секунду проходило бы не 50, а 60 прерываний работы процессора, т. к. там частота сети равна 60 Гц[1]).

Для того, чтобы исполнить паузу на m пятидесятых долей секунды, Вам достаточно загрузить это число m в регистровую пару BC, а затем вызвать процедуру ПЗУ PAUSE_1, расположенную по адресу 1F3DH (7997). Работая, процедура обеспечит заданную паузу. Во время своей работы процедура регулярно опрашивает системную переменную FLAGS (5C3BH = 23611). Ее пятый бит, если он включен, свидетельствует о том, что произошло нажатие клавиши. Таким образом, пауза может быть прервана нажатием любой клавиши - все, как в стандартном БЕЙСИКе. Если Вы зашлете в регистровую пару BC ноль, то установленная пауза будет неограниченной, то есть до нажатия клавиши.

Вам, может быть, потребуется случай, когда пауза должна быть выдержана в течение заданного времени и не должна прерываться нажатием клавиши, поможет несложная процедура (листинг 6).

Зашлите в BC требуемую величину задержки и вызывайте эту процедуру.

Листинг 6.

```
78     PAUSE      LD A, B      ; Проверка BC на
B1                OR C         ; ноль.
C8                RET Z        ; Выход, если так.
0B                DEC BC       ; Уменьшение BC,
76                HALT         ; Задержка на 1/50
                               ; секунды (точнее - до
                               ; следующего прерывания,
                               ; которое пройдет
                               ; чуть быстрее).
18F9              JR PAUSE     ; Возврат для повтора.
```

Рисование точек.

Для изображения точек на экране из машинного кода есть два приема. Первый, и наиболее простой состоит в том, чтобы загрузить координату X в регистр C, а координату Y - в регистр B, а затем вызвать процедуру PLOT_SUB, расположенную по адресу 22E5H (8933). Если у Вас координаты позиции печати точки уже содержатся в системной переменной COORDS (5C7D = 23677), то входить в эту процедуру можно в точке 22E8H (8936). Обратите внимание на то, что когда процедуры ПЗУ вычисляют по координатам позиции печати адрес соответствующего байта в дисплейной памяти, они коррумпируют регистр HL. Если он Вам нужен, его надо сохранить на стеке и потом восстановить.

Другой метод может быть более приемлем для тех энтузиастов, которые работают со встроенным калькулятором. Обе координаты должны быть предварительно помещены на вершину стека калькулятора в порядке: сначала X, затем Y. Вызовом процедуры PLOT (22DCH = 8924) эти координаты снимаются со стека и точка печатается на экране.

---

[1] Это связано частотой кадровой развертки видеосигнала формата PAL, принятого в Великобритании (50 Гц). (Прим. OCR)

Рисование прямых линий.

Здесь также существуют два метода работы из машинного кода, но прежде, чем мы их рассмотрим, напомним, что в операторе БЕЙСИКа DRAW X, Y параметры X и Y не являются координатами экрана, а являются "смещением" точки конца отрезка относительно точки его начала. "Смещение" может быть не только положительным, но и отрицательным.

Метод 1.

"Смещение" X загружается в регистры C и E. При этом в регистре C устанавливается его абсолютная величина ABS (X), а в регистре E устанавливается 01, если "смещение" положительно или равно нулю, но FFH, если отрицательно. Аналогично "смещение" Y загружается в регистры B и D. В регистре B - ABS (Y), а в регистре D - 01 или FFH.

После этого можно рисовать линию, вызвав процедуру DRAW_LINE с точкой входа 24BAH (9402). Линия будет нарисована, но следует предусмотреть сохранение регистровой пары H'L' (альтернативная), т.к. она будет коррумпирована. Второй путь, как и для печати точек, состоит в использовании калькулятора. Смещения X и Y (в таком порядке) должны быть установлены на вершине стека калькулятора и тогда та же процедура DRAW_LINE, но с точкой входа 24B7 (9399) нарисует отрезок прямой на экране. H'L' здесь коррумпируется точно так же.

Рисование дуги.

Оператор Бейсика DRAW может вычерчивать между двумя точками не только прямые линии, но и соединять их с помощью дуги. Форма оператора - DRAW X,Y,A. Параметр A определяет угловую меру дуги, а X, Y - "смещения". Угловая мера "A" задается в радианах.

Чтобы выполнить аналогичную задачу из машинного кода, следует все три параметра поместить на стек встроенного калькулятора в порядке X, Y, A. Затем вызывается процедура DRAW_ARC (2394H = 9108). Дуга рисуется, как серия очень маленьких прямых, т.е. эта процедура в своей работе многократно вызывает процедуру DRAW_LINE и, следовательно, регистровая пара H'L' (альтернативные) также неизбежно коррумпируется.

Рисование окружности.

Здесь тоже ведется работа через встроенный калькулятор. Координаты центра окружности X и Y, а затем радиус R должны быть помещены на вершину стека калькулятора, а затем выполняется вход в процедуру CIRCLE через точку входа 232DH (9005) для изображения окружности. H'L'- коррумпируется.

Проверка точки экрана.

С помощью функции POINT (X,Y) Вы можете проверить включен или выключен пиксел экрана с координатами X, Y. Для имитации работы этой функции из машинного кода у Вас есть два пути. Во-первых, Вы можете прогрузить регистры B и C координатами X и Y, после чего вызвать процедуру POINT_SUB с точкой входа (22CEH = 8910), а во-вторых, можете установить X и Y на вершине стека калькулятора и вызвать эту же процедуру с точкой входа 22CBH (8907). И в том и в другом случае результат будет оставлен на вершине стека. Снять его оттуда можно, обратившись к процедуре ПЗУ FP_TO_A (2DD5H = 11733). Эта процедура перешлет результат в регистр A процессора. Если пиксел выключен, то есть имеет цвет PAPER, Вы получите 0, а если он включен, т. е. имеет цвет INK, Вы получите единицу.

Проверка атрибутов экрана.

С помощью функции ATTR (Y,X) Вы можете проверить установку атрибутов в заданном знакоместе с координатами Y, X. Здесь X -номер экранного ряда (1...24), а Y - номер столбца (1...32). Для имитации работы этой функции из машинного кода Вы можете загрузить регистры B и C координатами Y и X, после чего вызвать процедуру S_ATTR_S с точкой входа (2583H = 9603), а во-вторых, можете установить Y и X на вершине стека калькулятора и вызвать эту же процедуру с точкой входа 2580H (9600). И в том, и в другом случае результат будет оставлен на вершине стека, снять его оттуда можно, обратившись к процедуре ПЗУ FP_TO_A (2DD5H = 11733), которая перешлет результат в аккумулятор. Анализируя данные по битам, Вы сможете установить параметры цветовых атрибутов в заданном знакоместе.

Раскладку по битам см. на рис. 3.

Проверка содержимого заданного знакоместа.

В БЕЙСИКе вы можете воспользоваться функцией SCREEN$ (Y,X) для того, чтобы определить, есть ли какой-нибудь символ ASCII в данном знакоместе с координатами Y и X и если есть, то что это за символ. То есть, эта функция может исполнять сканирование экрана. Надо, правда, сказать, что работа этой функции в БЕЙСИКе не отличается надежностью. В работе активно используется стек калькулятора, на котором может образовываться такая путаница, что например IF STRING$ (0,0)=STRING$(0, 1) THEN PRINT "TRUE" будет всегда выдавать "TRUE" независимо от того, что содержится в знакоместах (0,0) и (0,1).

К счастью, мы можем организовать сканирование и из машинного кода, причем там такой путаницы не будет. Вы можете использовать процедуру ПЗУ S_SCRN$_S либо с точкой входа 2535H (9525), если Вы поместили Y и X на вершину стека калькулятора (именно в таком порядке), либо с точкой входа 2538H (9528), если вы поместили в регистр C номер экранного ряда, а в B - номер столбца.

И в том, и в другом случае результат будет оставлен на вершине стека калькулятора. Чтобы переслать его оттуда в регистры микропроцессора, воспользуйтесь процедурой FP_TO_AEDCB (2BF1H = 11249). После этого в BC будет содержаться 0, если такого символа в наборе ASCII не существует или 1, если символ опознан. Узнать, что это за символ, можно посмотрев содержимое аккумулятора - там будет содержаться его код.

Конечно, в результате сканирования смогут быть разысканы только символы ASCII - от 20H ("пробел") до 7FH ("копирайт"), т.к. только они имеются в таблице шрифта, на которую указывает системная переменная CHARS (23606 -5C36H), но это дело можно доработать. Мы не приводим здесь процедуру, которая способна просканировать весь экран и определить как символы ASCII, так и символы графики пользователя и символы блочной графики в связи с нехваткой места. Будущие читатели смогут познакомиться и с ней и с многими другими интересными и полезными вещами на страницах нашей новой книги.

СЛУЧАЙНАЯ ГРАФИКА

(Глава из книги "Дизайн Ваших программ")

Продолжая разговор о графике "Спектрума", мы представляем отрывок из готовящейся сейчас к изданию книги "Персональный компьютер СПЕКТРУМ. Дизайн ваших программ. " Это заключительный (четвертый) том готовящейся к выпуску серии, посвященной графике.

По мере окончания подготовки, книга будет предложена к изданию по регионам. Следите за нашей информацией.

СЛУЧАЙНАЯ ГРАФИКА

Есть один аспект, связанный с 48-килобайтными "Спектрумами", в котором проявляется ограниченность их ресурсов оперативной памяти - это графика. Здесь ограничения более, чем очевидны. Графические изображения могут чудовищно расходовать память Вашей машины.

Вам никогда не приходилось задумываться, почему одни программы имеют богатую графику, а другие - выглядят серыми и невыразительными? Что, может быть в первом случае работал хороший художник, а во втором - плохой? Да, конечно, для любительских программ это вполне справедливо, но если речь идет о крупной фирме, способной привлечь к работе и оплатить труд первоклассных дизайнеров, то ясно, что не в художнике дело. Он бы Вам изобразил все, что хотите, а вот как разместить все его идеи в ограниченном объеме памяти?!

Итак, графический дизайн программы оказывается тесно увязан с программными возможностями, с общей концепцией проекта, и насколько хороша будет программа, зависит в конечном итоге не только от художника, а от совместных усилий всей команды в целом, начиная с руководителя проекта и кончая самым последним программистом. Вот почему так важно еще на этапе предварительной проработки концепции будущей программы четко определиться с потребностями в графике и наметить для себя те программистские приемы и методы, которыми эти потребности будут обеспечиваться.

Сегодня мы Вас и познакомим с одним из таких приемов. Он состоит в том, чтобы предоставить генерацию графики самому компьютеру по некоторому заданному алгоритму, поставив тем самым производство графических изображений "на поток". У этого метода есть существенный недостаток, состоящий в том, что многие изображения могут оказаться "похожими" друг на друга и различаться лишь в деталях, но это уже вопрос баланса программы. Вы ведь можете использовать полученную таким образом графику только в качестве фоновой, накладывая на нее незначительные по размерам и по объемам расходуемой памяти графические объекты.

Такая техника применяется, как правило, в адвентюрных программах, поскольку именно в них приходится иметь дело с огромным количеством иллюстраций (например 32000 - в программах фирмы BEYOND - "Sorderon's Shadow" или, скажем, "Lords of Midnight", хотя есть и много более внушительные примеры). По всей видимости, первой применила такую технику фирма LEGEND в своей знаменитой программе "Valhalla".

Теперь давайте перейдем к конкретной задаче и посмотрим, как используя "спектрумовский" генератор случайных чисел, можно получить образцы такой "фоновой" графики (ее еще называют ландшафтной графикой) для Ваших программ.

Мы сразу должны оговориться, что полное исследование всех возможностей предлагаемого метода может занять у вас не один месяц и полностью осветить все варианты в рамках одной книги нет никакой возможности. Мы будем просто руководствоваться генеральным принципом "ИНФОРКОМа" - научить приемам и методикам, а скрупулезное исследование оставляем читателю на самостоятельную проработку.

Рассмотрим в качестве примера следующую Бейсик-программу. Не обращайте пока внимания на то, что она очень медленно работает, сейчас для понимания самой сути идеи нам это не так важно (Листинг 1).

ЛИСТИНГ 1.

```
1 DEF FN r(x)=INT(RND*x)
2 BORDER 0: INK 0: LET n=1: GO TO 1000
7 REM
8 REM ** Рисуем рамку и наносим фоновые цвета **
9 REM
10 CLS: PLOT 0,95: DRAW 128, 0: DRAW 0,80: PRINT AT 1, 17; "Рисунок номер..."; AT 3, 24; n;
      AT 0,0;
15 PRINT PAPER 5,',',',',',' PAPER 1, ',' PAPER 6, ',',': RETURN
17 REM
18 REM ** Рисуем горы **
19 REM
20 INK 0:LET ht =FN r(24): LET mx=23: LET mn=0: LET p=FN r(2)
25 FOR i=0 TO 127: PLOT i,136: DRAW 0, 7+ht
30 IF p THEN LET ht=ht+FN r(2):IF ht>= mx THEN LET P=0: LET ht=mx: LET mn=1+FN r(7) : GO TO
      50
40 IF NOT p THEN LET ht=ht-FN r(2): IF ht<=mn THEN LET p=1: LET ht=mn:LET mx=8+FN r(16)
50 NEXT i: RETURN
77 REM
78 REM ** Рисуем озеро **
79 REM
80 INK 7: FOR i=0 TO 49
90 LET x = FN r(125): LET y=128+ FN r(8): PLOT x,y
100 DRAW FN r(4),0: NEXT i: RETURN
137 REM
138 REM ** Рисуем тростник **
139 REM
140 INK 4: FOR i=0 TO 127: LET mn=FN r(2): LET mx=1 +FN r (7)
150 PLOT i,119+mn: DRAW 0, mx-mn: NEXT i : RETURN
177 REM
178 REM ** Рисуем каменистую равнину **
179 REM
180 INK 0: FOR i=0 TO 49
190 LET x=FN r(128): LET y=104+FN r(8):PLOT x,y: NEXT i: RETURN
237 REM
238 REM ** Рисуем траву на переднем плане **
239 REM
240 INK 4: FOR i=0 TO 127: PLOT i,96: DRAW 0, FN r(8): NEXT i: RETURN
997 REM
998 REM ** Главный цикл рисования картинки **
999 REM
1000 CLS: RANDOMIZE n: GO SUB 10: REM ** Очистка картинки **
1005 GO SUB 20: REM ** Рисуем горы **
1010 GO SUB 80: REM ** Рисуем озеро **
1015 GO SUB 140: REM ** Рисуем ТРОСТНИК **
1020 GO SUB 160: REM ** Рисуем почву **
1025 GO SUB 240: REM ** рисуем траву **
1030 INK 0: PRINT #1; AT 1,9; FLASH 1; "Еще рисунок?"
1040 PAUSE 0: LET n=n+1: GO TO 1000
```

Как Вы видите, эта небольшая программа использует функцию генерации случайных чисел компьютера RND, с помощью которой случайным образом задаются параметры графических операторов БЕЙСИКа DRAW и PLOT. Программа довольно эффектно рисует пейзажи, содержащие горы, воду, почву и т. п. Но что самое интересное для программиста - так это то, что функция RND, как Вы должно быть знаете, вовсе не выдает случайные числа, а только лишь псевдо-случайные. То есть, имеется некоторая заранее предопределенная последовательность, из которой эти псевдослучайные числа и выбираются. Эта последовательность рассчитывается с помощью небольшой процедуры, имеющейся в ПЗУ

компьютера.

В качестве исходного параметра при работе этой процедуры используется значение системной переменной SEED, содержащейся в двух байтах по адресам 23670, 23671 (5C76H). Изменить содержимое этой системной переменной и начать генерацию новой псевдослучайной последовательности можно оператором RANDOMIZE n, что вы и видите в программе в строке 1000. В качестве параметра n может быть любое число от 0 до 65535, т.е. таким образом, Ваш генератор случайных чисел может выдать псевдослучайную последовательность из 65536-ти чисел, после чего они начнут повторяться (последовательность вырождается).

Если Вы зададите в операторе RANDOMIZE в качестве параметра n например номер картинки, то Вы всегда для одного и того же, номера будете иметь один и тот же пейзаж, но для разных n они будут различны.

Конечно, можно пойти еще дальше и, в зависимости от n, вызывать те или иные процедуры, которые внесут дополнительное разнообразие в Ваш рисунок, например, можно наложить какие-то небольшие рисунки на Ваш фоновый пейзаж.

Несложной заменой параметра цвета INK озеро, изображаемое на среднем плане, может быть преобразовано в болото, в снежный покров, в песчаную пустыню.

Проблемы скорости работы.

К сожалению, перед нами по-прежнему стоит проблема, связанная с медленной работой вышеприведенной программы. На первый взгляд, все это не так сложно, надо только заменить медленно работающие операторы PLOT, DRAW и RND на вызов из машинного кода тех процедур ПЗУ, которые им соответствуют, но результат будет ненамного лучше. Дело в том, что стандартная процедура ПЗУ, выполнявшая функцию RND, работает настолько медленно, что от того, что вы ускорите обращение к ней применением машинного кода, Вы мало чего достигнете. Надо искать другое решение.

Во-первых, честно говоря, нам и не нужна очень случайная последовательность для нашей задачи и нет смысла проходить все те манипуляции с числами, которые выполняет процедура ПЗУ.

Во-вторых, зачем нам 65536 случайных чисел? Может быть, на первый случай будет достаточно и 256-ти?

Ну, и если это все так, то мы сами можем создать где-то в памяти небольшую таблицу из 256 чисел, предварительно заполнить случайными числами, а затем брать их оттуда в готовом виде, не привлекая для этого обращение к ПЗУ.

Резервируем для этого область памяти, начиная с адреса 64744:

```
FOR i = 64744 TO 64799:
POKE i, INT(RND*255)): NEXT i
```

Теперь представьте, что у Вас есть некий указатель, который указывает на какой-то пункт в этой таблице. Предположим, вам понадобилось некоторое случайное число - Вы возьмете его из того пункта, на который указывает указатель, а сам указатель передвинете на следующую позицию для последующего использования. Когда указатель дойдет до конца таблицы, его совсем несложно опять перегнать в ее начало, так мы сможем обеспечить 256 различных картинок в своей программе, различающихся начальной позицией указателя. Конечно, при рисовании только одной картинки, указатель может много раз пробежать по таблице, но в этом нет никакой беды, потому что если например при рисовании травы и возникнет некое подобие регулярности, то оно не будет выглядеть неестественно, а графика, с помощью которой изображены горы, достаточно сложна, чтобы заметить в ней некоторые внутренние повторения. Для проведения математических или статистических исследований, такой подход был бы совершенно неприемлем, но для графики он вполне годится.

Трансляция БЕЙСИКа в машинный код.

Читатель, который интересуется только идеями, приемами и методами, мог бы на этом и завершить чтение данной главы и приступить к самостоятельным исследованиям по созданию образцов случайной машинной графики. Те же, кому необходим инструмент с

достойными рабочими характеристиками, получат ниже рекомендации о том, как значительно ускорить работу алгоритма, подключив в дело машинный код.

Прежде всего, для работы нам необходимы четыре процедуры:

- рисования точки (аналог PLOT);
- рисования линии (аналог DRAW);
- генерации случайных чисел (аналог RND);
- процедура для подготовки экрана (очистка экрана, рисование рамки, предварительное окрашивание фона в цвета PAPER) - назовем ее BCGRND от слова BackGround - фон, задний план.

PLOT.

С этой процедурой все просто. Ее даже не надо делать, вполне можно использовать процедуру, имеющуюся в ПЗУ по адресу 22E5H (8933), вызвав ее командой процессора CALL 22E5H.

Надо только предварительно перед вызовом этой процедуры выставить ее параметры в регистрах B и C микропроцессора. Так, если Вам надо выполнить PLOT x,y, то в регистр C должно быть загружено число "x", а в регистр B - число "y".

DRAW.

Здесь мы поступим аналогично, воспользовавшись процедурой ПЗУ, служащей для рисования линий и расположенной по адресу 24BAH (9403). Параметры "x" и "y" оператора DRAW x,y выставляются предварительно в регистрах C и B совершенно аналогично, но надо еще выставить в регистрах D и E знаки "y" и "x", соответственно. Во время своей работы, процедура "портит" содержимое регистровой пары HL регистров альтернативного набора. Поэтому, чтобы не было никаких коллизий с прочими процедурами, необходимо в общем случае запоминать содержимое этой пары на стеке процессора, а после окончания работы процедуры - восстанавливать его.

RND.

Работа этой процедуры представляет для нас несколько больший интерес. Нам надо, чтобы после ее вызова, она выдавала бы эквивалент того, что выдает оператор INT(RND*x), где "x" - число, предварительно загруженное в аккумулятор процессора. Результат работы процедуры при выходе из нее остается там же - в аккумуляторе. Одним словом, эта процедура должна делать то же самое, что и функция FN r(), определенная нами ранее в строке 1 вышеприведенной БЕЙСИК-программы.

Для работы этой процедуры Вам надо также предварительно заполнить 256-байтную таблицу псевдослучайных чисел, как это было показано выше. Сделайте это сами, а впоследствии мы сможем отгружать таблицу вместе с машинным кодом наших процедур единым блоком.

Обеспечивать задание начальной точки нашей псевдослучайной последовательности мы сможем, изменяя при помощи POKE значение переменной POINT, находящейся по адресу 65057. Так, заслав сюда номер Вашего рисунка, Вы получите его на экране.

BCGRND

Процедура служит для подготовки левой верхней четверти экрана к рисованию вашей картинки. Это оставляет еще достаточно места для текстовых сообщений, если Вы пишете адвентюрную или обучающую программу.

Процедура достаточно проста для тех, кто хоть немного знаком с машинным кодом. Для тех же, кто делает первые шаги, укажем только, что сначала она вызывает из ПЗУ процедуры PLOT и DRAW для рисования рамки, а затем приступает к закрашиванию фона цветами PAPER. Для неба - это 5 рядов цвета PAP_1 (в нашем случае это голубой), для озера - 2 ряда PAP_2 (синий), а для почвы – три ряда PAP_3 (желтый). Закрашивание выполняется командой процессора RST 16, служащей для вывода информации на экран, закрашивание рядов выполняется в цикле, окончание цикла задано командами процессора DJNZ.

Возможно, стоит пояснить, как происходит закрашивание каждого ряда, в принципе, чтобы окрасить ряд из 16-ти знакомест в цвет PAPER, надо напечатать 16 пробелов, но это расточительная операция. Здесь все сделано изящнее с помощью кода управления печатью

CHR 6 (строки 730, 830, 930). Этот управляющий символ называется PRINT COMMA - "Запятая Оператора PRINT". Его действие эквивалентно действию оператора TAB 16, т.е. он выводит на экран те самые нужные нам 16 пробелов.

Окончание заполнения ряда и переход к новому ряду выполняются печатью 13-го символа (CHR 13 = "ENTER") - строки 759, 850, 950.

Процедура организована таким образом, что Вы всегда имеете возможность поменять значения фоновых цветов PAP_1, PAP_2, PAP_3. Они заданы директивой АССЕМБЛЕРа DEFB в ячейках памяти 65137, 65138, 65139.

Распечатка этих четырех вспомогательных процедур приведена в Листинге_2. Наберите их с помощью любого доступного Вам АССЕМБЛЕРа и откомпилируйте в машинный код, начиная с адреса 65000 по 65139. Если Вы не забыли предварительно составить таблицу псевдослучайных чисел в адресах с 64744 по 64799, то можете выгрузить их совместно.

Листинг 2.

```
       10   ; Распечатка четырех
       20   ; вспомогательных процедур
       30   ; DRAW, PLOT, RND и BCGRND
65000  40               ORG 65000
       50   DRAW
       60   ; Процедура эквивалентна DRAW C, B
65000  70               EXX
65001  80               PUSH HL
65002  90               EXX
65003  100              LD DE,0101H
65006  110              CALL 24BAH
65009  120              EXX
65010  130              POP HL
65011  140              EXX
65012  150              RET
       160
       170
       180
       190  ; процедура эквивалентна PLOT C, B
       200  PLOT         EQU 22E5H
       210
       220
       230  RAND
       240  ; Процедура принимает из аккумулятора число
       250  ; и замещает его на результат вычисления
       260  ; INT (RND*A). Таблица случайных чисел в
       265  ; адресах 64741...64999 должна быть заполнена.
65013  270              LD (VALUE),A       ; Запомнили содержимое аккумулятора в ячейке
                                           ; с меткой VALUE.
65016  280              LD HL,64744        ; Начало таблицы случайных чисел.
65019  290              LD DE,(POINT)      ; Ввели текущее значение указателя.
65023  300              LD D,0
65023  310              ADD HL, DE         ; Встали на позицию указателя.
65026  330              LD A,E             ; Ввели значение указателя.
65027  330              INC A              ; Передвинули указатель,
65028  340              LD (POINT),A       ; Запомнили новое положение указателя.
65031  350              LD A, (HL)         , Приняли случайное число.
65032  360              CALL 2D28H         ; Эта процедура ПЗУ выполняет
                                           ; сразу две полезные функции -
                                           ; переводит число в действительную
                                           ; форму (с плавающей точкой)
                                           ; и помещает его на стек калькулятора.
65055  370              LD A,255H
65037  380              CALL 2D28H         ; Поместили на стек 255,
65040  390              RST 40             ; Включение калькулятора,
65041  400              DEFB 5             ; Команда калькулятора
                                           ; "деление". На стеке
```

```
                                        ; осталось "RND".
65042 410               DEFB 56         ; Выключение калькулятора.
65043 420               LD A,(VALUE)
66046 430               CALL 2D28H      ; Поместили на стек значе-
                                        ; ние VALUE.
65049 440               RST 40          ; Включение калькулятора,
65050 450               DEFB 4          ; Команда калькулятора
                                        ; "умножение". На стеке
                                        ; получено RND*VALUE.
65051 460               DEFB 39         ; Команда калькулятора INT.
65052 470               DEFB 56         ; Выключение калькулятора.
66053 480               CALL 2DD5H      ; очень важная процедура
                                        ; ПЗУ. Снимает число с вер-
                                        ; шины стека калькулятора и
                                        ; пересылает его в регистр A
                                        ; микропроцессора.
65050 490               RET
65057 500  POINT        DEFB 0          ; переменная POINT.
65058 510  VALUE        DEFB 0          ; Переменная VALUE.
      520
      530
      540  BCGRND
      550                               ; Процедура рисует рамку и фоновые полосы в
      560                               ; цвете PAPER, заданном в PAP_1, PAP_2, PAP_3.
65059 570               LD A, 2
65061 580               CALL 1601H      ; Вызовом этой процедуры
                                        ; выбирается устройство вывода
                                        ; информации. Поскольку
                                        ; в аккумуляторе установлено
                                        : число 2, вывод будет
                                        ; идти на экран.
65062 590               LD B,95
65064 600               LD C,0
65068 610               CALL PLOT
65071 620               LD B,0
65073 630               LD C,128
65075 640               CALL DRAW
65078 650               LD B,80
65080 660               LD C,0
65082 670               CALL DRAW
66085 680               LD A,17         ; Код PAPER.
65087 690               RST 16
65088 700               LD A,(PAP_1)    ; Установка цвета PAPER.
65091 710               RST 16
65092 720               LD B,5          ; 5 рядов - небо.
65094 730  LOOP1        LD A,6          ; Управляющий код CHR 6.
65096 740               RST 16
66097 750               LD A,13         ; Управляющий код ENTER.
65099 760               RST 16
66100 770               DJNZ LOOP1
65102 780               LD A,17
65104 790               RST 16
65105 800               LD A,(PAP_2)
65108 810               RST 16
65109 820               LD B,2          ; 2 ряда - озеро
65111 830  LOOP2        LD A,6
65113 840               RST 16
65114 850               LD A,13
66116 860               RST 16
65117 870               DJNZ LOOP2
65119 880               LD A,17
65521 890               RST 16
65122 900               LD A,(PAP_3)
65125 910               RST 16
65126 920               LD B,3          ; 3 ряда - земля
```

```
65128  930   LOOP3       LD A,6
65130  940               RST 16
65131  950               LD A,13
65133  960               RST 16
65134  970               DJNZ LOOP3
65136  980               RET
65137  990   PAP_1       DEFB 5            ; - голубой
65138  1000  PAP_2       DEFB 1            ; - синий
65139  1010  PAP_3       DEFB 6            ; - желтый
```

Изменив значение по адресу 65137 на 1, Вы вместо голубого неба получите синее и день превратится в ночь. А замена числа 1 на 4 по адресу 65138 превратит озеро в болото. Это ваши дополнительные резервы по созданию графики несложной по исполнению, малоемкой по расходуемой памяти, но способной внести своеобразную атмосферу в любую программу.

Теперь нам осталось воспользоваться созданными процедурами и довести дело до конца, написав программу, которая будет нам генерировать такие картинки в огромных количествах в доли секунды.

Распечатка представлена в Листинге_3. Он не снабжен подробными комментариями, поскольку в нем сделано все возможное, чтобы максимально следовать той БЕЙСИК-программе, с которой мы начинали. Она фактически и является комментарием этой процедуры.

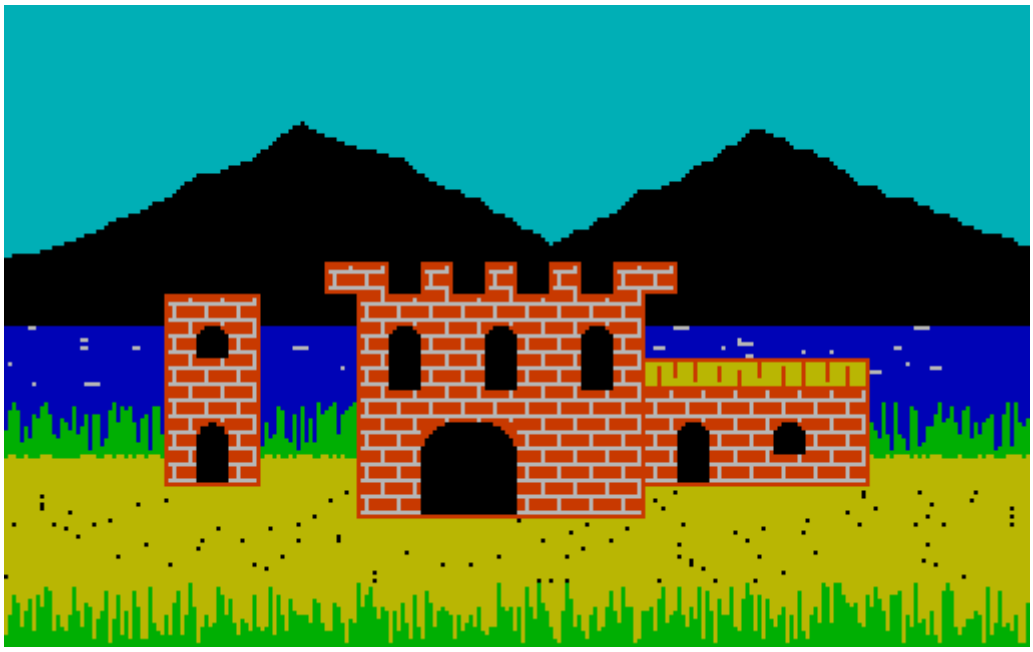
Листинг_3.

```
65097  750               LD A,13
                         ORG 65140
       1020
       1030  MOUNT       ; Горы.
65140  1040              CALL INIT
65143  1050              LD A,24
65145  1060              CALL RAND
65148  1070              LD (HT),A
65151  1060              LD A,23
65153  1090              LD (MX),A
65156  1100              XOR A
65157  1110              LD (MN),A
65160  1120              LD A,2
65162  1130              CALL RAND
65165  1140              LD (PP),A
65168  1150  LOOP4       LD A,(II)
65171  1160              LD C,A
65179  1170              LD B,136
65174  1180              CALL PLOT
65177  1190              XOR A
65178  1200              LD C,A
65179  1210              LD A,(HT)
65182  1220              ADD A,7
65184  1230              LD B, A
65185  1240              CALL DRAW
65188  1250              LD A,(PP)
65191  1260              CP 0
65193  1270              JP NZ,UP
65196  1280              LD A,2
65198  1290              CALL RAND
65201  1300              LD B,A
65802  1310              LD A,(HT)
65205  1320              SUB B
65206  1330              LD (HT),A
65209  1340              LD B,A
65310  1350              LD A,(MN)
65213  1360              CP B
65214  1370              CALL NC,SWAP2
```

```
65217 1380 CONT         LD A,(II)
65220 1390              INC A
65221 1400              CP 128
65223 1410              RET Z
65224 1420              LD (II),A
65227 1430              JP LOOP4
65230 1440 UP           LD A,2
65232 1450              CALL RAND
65235 1460              LD B,A
65236 1470              LD A,(HT)
65239 1480              ADD A,B
65240 1490              LD (HT),A
65243 1500              LD A,(MX)
65246 1510              LD B,A
65247 1520              LD A,(HT)
65250 1530              CP B
65251 1540              CALL NC,SWAP1
65254 1550              JP CONT
65257 1560 SWAP1        LD A,B
65258 1570              LD (HT),A
65261 1580              LD A,7
65263 1590              CALL RAND
65266 1600              ADD A,1
65268 1610              LD (MN),A
65271 1620              XOR A
65272 1630              LD (PP),A
65275 1640              RET
65276 1650 SWAP2        LD (NT),A
65279 1660              LD A,16
65281 1670              CALL RAND
65284 1680              ADD A,8
65286 1690              LD (MX),A
65289 1700              LD A,1
65291 1710              LD (PP),A
65294 1720              RET
65295 1730 II           DEFB 0
65296 1740 PP           DEFB 0
65297 1750 HT           DEFB 0
65296 1760 MX           DEFB 0
65299 1770 MN           DEFB 0
      1780
      1790
      1800 LAKE         ; Озеро.
65300 1810              CALL INIT
65303 1620 LOOP5        LD A,125
65305 1630              CALL RAND
65308 1840              LD C,A
65309 1850              LD A,8
65311 1660              PUSH BC
65312 1870              CALL RAND
65315 1880              POP BC
65316 1890              ADD A,128
65318 1900              LD B,A
65319 1910              CALL PLOT
65322 1920              LD A,4
65324 1930              CALL RAND
65327 1940              LD C,A
65326 1950              LD B,0
65330 1960              CALL DRAW
65333 1970              LD A,(II)
65336 1980              INC A
65337 1990              CP 50
65339 2000              RET Z
65340 2010              LD (II),A
65343 2020              JP LOOP5
```

```
       2030
       2040
       2050 REEDS        ; Тростник
65346  2060              CALL INIT
65349  2070 LOOP6        LD A,2
65351  2080              CALL RAND
65354  2090              LD (MN),A
65357  2100              LD A,7
65359  2110              CALL RAND
65362  2180              ADD A,1
65364  2130              LD (MX),A
65367  2540              LD A,(II)
65370  2150              LD C,A
65371  2160              LD A, (MN)
65374  2170              ADD A,119
65376  2180              LD B,A
65377  2190              CALL PLOT
65380  2200              XOR A
65381  2210              LD C,A
65382  2220              LD A,(MN)
65385  2230              LD B,A
65366  2240              LD A,(MX)
65389  2250              SUB B
65390  2260              LD B,A
65391  2270              CALL DRAW
65394  2280              LD A,(II)
65397  2290              INC A
65398  2300              CP 128
65400  2310              RET Z
65401  2320              LD (II),A
65404  2330              JP LOOP6
       2340
       2350
       2360 GROUND       ; Земля.
65407  2370              CALL INIT
65410  3380 LOOP7        LD A,128
65412  2390              CALL RAND
65415  2400              LD C,A
65416  2410              LD A,8
65416  2420              PUSH BC
65419  2430              CALL RAND
65422  2440              POP BC
65423  2450              ADD A,104
65425  2460              LD B,A
65426  2470              CALL PLOT
65429  2480              LD A,(II)
65432  2490              INC A
65433  2500              CP 50
65435  2510              RET Z
65436  2520              LD (II),A
65439  2530              JP LOOP7
       2540
       2550
       2560 GRASS        ; Трава.
65442  2570              CALL INIT
65445  2580 LOOP8        LD A,(II)
65446  2590              LD C,A
65449  2600              LD B,96
65451  2610              CALL PLOT
65454  2620              LD A,8
65456  2630              CALL RAND
65459  2640              LD B,A
65460  2650              LD C,0
65462  2660              CALL DRAW
65465  2670              LD A,(II)
```

```
65468  2680              INC A
65469  2690              CP 128
65471  2700              RET Z
65472  2710              LD (II),A
65475  2720              JP LOOP8
       2730
       2740
       2750  INIT        ; процедура инициализации.
       2760              : Устанавливаем цвет PAPER
                         ; равный transparent -
                         ; то же самое, что
                         ; и PAPER 8.
65478  2770              LD A,248
65480  2780              LD (MASKT),A
       2790              ; Обнуление счетчика.
65483  2800              XOR A
65484  2810              LD (II),A
65487  2820              RET
       2830
       2840  MASKT       EQU 23696
       2850  PLOT        EQU 8933
       2850  DRAW        EQU 65000
```



Пример применения случайной графики.
Замок на переднем плане наложен более поздно.

После того, как Вы ассемблируете процедуры, приведенные в листингах, не забыв, конечно и о таблице случайных чисел, Вы принципе уже готовы к тому, чтобы начать рисовать. Дело осталось только совсем за малым: нужна программа-драйвер, которая объединит все части в единое целое и будет ими управлять. Ее можно сделать и на БЕЙСИКе (См. Листинг_4). Приведенный пример обеспечит Вам генерацию 256 различных картинок и прокрутку их одну за другой по нажатию клавиши.

Попробуйте, и Вы увидите, что скорость воспроизведения графики вполне достаточна для того, чтобы использовать ее в реальных программах и уж совсем несравнима с тем, что мы имели в БЕЙСИКе.

Практически мы с Вами затратили на генерацию 256-ти картинок около 800 байтов (вместе с таблицей случайных чисел), т.е. примерно по 3 байта на картинку. Это нормальное соотношение, но его можно было бы еще улучшить в несколько раз.

Во-первых, ясно, что если бы мы делали большее количество рисунков, то оно было бы лучше.

Во-вторых, хранение в памяти таблиц - это наиболее простой в смысле понятности способ и годится только для демонстрации вследствие своей расточительности. Профессионалы же пишут сами краткие и устойчивые алгоритмы генерации псевдослучайных последовательностей.

В третьих, если Вы проанализируете листинг 2, то увидите, насколько же мало байтов уходит на то, чтобы смоделировать тот или иной графический объект (горы, озеро, луг, грунт и т.п.). Пользуясь предложенной идеей, Вы всегда сможете развить число этих объектов (крепостная стена, изгородь, заросли кустарника, лес, море, скалы, звездное небо, космические тела и многое другое).

Ну и, наконец, в четвертых, мы уже говорили о том, что большое разнообразие при смысловой содержании и сохранении духа и атмосферы программы Вы сможете получить, накладывая на полученные пейзажи некрупные графические объекты, не занимающие большого места в памяти. Одним из рациональных приемов при этом является формирование их на основе блочной графики UDG.

ЛИСТИНГ_4.

```
1 REM ** БЕЙСИК-драйвер **
2 REM
3 BORDER 0: CLEAR 64743: LOAD ""CODE 65000
4 REM **Заполнение псевдослучайной таблицы**
5 REM
6 FOR i=64744 TO 64999: POKE i, INT(255*RND): NEXT i
7 LET n=0: REM **Инициализация счетчика**
8 REM **Главный цикл**
9 REM
10 LET n=n+1: POKE 65057, n
15 INK 0: CLS: RANDOMIZE USR 65059
20 PRINT AT 1, 17; "Номер рисунка..." AT 3,24; n
30 INK 0: RANDOMIZE USR 65140: REM **Горы**
35 INK 0: RANDOMIZE USR 65300: REM **Озеро**
40 INK 0: RANDOMIZE USR 65346: REM **Тростник**
45 INK 0: RANDOMIZE USR 65407; REM **Земля**
50 INK 0: RANDOMIZE USR 65442: REM **Трава**
60 PRINT #1; AT 1,9; FLASH 1: "Еще рисунок?": PAUSE 0
70 GO TO 10
```

Может быть, Вам потребуется не один набор UDG и Вы организуете эти наборы в банки. Впрочем, об этом речь еще впереди.

По мере готовности книги мы оповестим всех, кто готов заняться ее изданием и распространением в своих регионах.

Следите за нашей информацией!

КРИБЕДЖ

(Глава из книги "Настольные игры своими руками")

Вашему внимания предлагается глава из готовящейся к изданию книги "Персональный компьютер СПЕКТРУМ. Настольные игры своими руками. " Мы надеемся, что данный отрывок дает представление о содержании книги, которая будет предложена к изданию в ближайшее время.

КРИБЕДЖ

Считается общепризнанным тот факт, что крибедж - самая интересная карточная игра для двух играющих. Есть, правда, версии и для трех игроков и даже для четырех, но это уже не то. Вчетвером можно найти игру и поинтереснее (тот же бридж, например), а вот если Вас только двое, то крибедж может Вам очень и очень понравиться.

Предполагают, что изобрел крибедж и дал ему название английский поэт, лорд Джон Саклинг (1609-1642). Впоследствии первые колонисты завезли игру в Америку, где она и получила свой расцвет.

В чем прелесть крибеджа? Дело в том, что здесь, как и в любой другой карточной игре, определенную роль играет везение, но не слишком большую. У опытного игрока всегда есть возможность доказать, что счастье - это хорошо, но голова на плечах - лучше. Своим успехом крибедж во многом обязан правилам, которые сложными пожалуй не назовешь, но многообразными - можно. Счет очков идет настолько динамично, что для игроков даже существует специальное приспособление - доска с отверстиями. Игроки по мере набора очков переставляют свои колышки из отверстия в отверстие, стараясь как можно быстрее пройти путь к концу доски.

Предлагая Вам самостоятельно набрать эту программу, мы должны предварительно посвятить вас в правила этой игры, в правила подсчета очков и дать основы стратегии.

В игре используется полная колода карт - 52 листа. Старшинство карт следующее (сверху вниз): К,Д,В,10, ... 3,2,Т. Туз - младшая карта, каждая карта имеет свое достоинство. Все фигуры и десятка - по 10 очков, прочие - по номиналу, а туз - 1 очко.

Вся игра состоит из геймов.

Гейм состоит из раундов. Гейм закончен, когда кто-то наберет 121 очко. Победителю в гейме засчитывается одно очко, если его противник набрал более 61 очка и два очка, если тот набрал меньше. О том, до какого счета по геймам Вы будете играть - договоритесь сами, обычно играют до 6 или до 12-ти.

Сдача в игре производится по очереди, сдающий имеет значительное преимущество, он ведет активную игру. Его противник как бы держит оборону, сдача происходит в следующем порядке:

1. Каждому сдаются по 6 карт.

2. Каждый оставляет себе по 4 карты и по две карты сбрасывает на стол рубашкой вверх, не показывая их противнику. В результате у каждого осталось по 4 карты и на столе образовалась еще одна рука с четырьмя картами - эту руку называют КРИБ. Она принадлежит сдавшему.

3. После сноса криба противник "срезает" колоду и сдающий открывает верхнюю (после срезки карту). Эту карту называют "стартер". Она остается на колоде, но условно принадлежит всем играющим, в том числе и крибу, т.е. они имеют для игры по 4 карты, но для зачетов очков - по пять карт вместе со стартером.

Очки в раунде добываются двумя путями.

Во-первых, можно взять очки с игры, в этом смысле и сдающий и его противник равны между собой.

Во-вторых, даются очки за те комбинации карт, которые образовались на Вашей руке после сброса двух карт и вскрытия стартера. Поскольку сдающий кроме своей руки владеет

еще и крибом, то те комбинации, которые окажутся в нем после розыгрыша раунда, прибавятся к его очкам.

Игра.

Сначала рассмотрим, как набираются очки с игры. Игра производится поочередным выкладыванием карт на стол. При этом нельзя, чтобы сумма выложенных карт превысила 31 очко.

Первый выкладывает любую свою карту несдававший. Затем кладет карту сдатчик. При этом:

- если полученная сумма равна 15, ему запишется 2 очка;
- если его карта равна карте предыдущего хода, т.е. образуется "пара", ему запишется 2 очка.

Ход переходит к несдававшему. Он может сделать:

- "пятнадцать" (2 очка)
- "пару"(2 очка)
- "секанс" (три карты подряд, например В,10,Д или 3,2,Т - 3 очка).

Ход переходит и так далее.

В последующие ходы могут возникать и более сложные комбинации:

- секансы из более чем трех карт оцениваются во столько очков, сколько карт в секансе.
- "пэр рояль" - три одинаковые карты подряд дают 6 очков, т. к. из них можно составить 3 разные пары;
- "двойной пэр рояль" - 4 карты одного достоинства - 12 очков (6 разных пар).

Так игроки ходят по очереди до тех пор, пока один из них не сможет сделать ход. Ведь выходить за пределы 31 очка нельзя. Тогда он объявляет "ХОД" и его противник получает себе очко. Теперь противник может, если у него есть мелкие карты, выкладывать их на стол, оставаясь в пределах 31 очка. Так можно выложить за ход и две и три карты подряд. Если ему удается в результате своего хода набрать ровно тридцать одно, ему запишется 2 очка.

Когда и второй играющий не сможет сделать хода, то с оставшимися на руках картами игроки начинают новую выкладку, первым кладет карту тот, кто первым пропустил ход.

Ход последней картой (из восьми) дает еще одно очко, а если к тому же в результате этого хода образовалось 31, то добавляются 2 дополнительных очка.

Теперь, когда все карты выложены, игроки начинают подсчитывать свои очки за комбинации, которые у них были на руках к началу игры. Начинающим для простоты можно разобрать выложенные карты со стола по рукам и начать считать. В игре с компьютером подсчет сделает за вас машина, если Вы этого пожелаете. Стартер считается пятой картой для каждой из трех рук.

В зачет идут следующие комбинации:

"Пятнадцать" - 2 очка.

"Пара" - 2 очка.

"Пэр рояль" - 6 очков.

"Двойной пэр рояль" - 12 очков.

"Секанс" - по числу карт.

"Флешь" - 4 или 5 очков.

Флешь - это 4 карты одной масти в руке, но не в крибе (4 очка). Если ту же масть имеет стартер - то 5 очков, за 4 карты одной масти в крибе не дается ничего, но если эту масть имеет и стартер, то хозяин криба (сдающий) получает 5 очков.

"Его благородие" - Если у вас на руках есть валет той же масти, что и масть стартера, вы получаете одно очко.

"Челядь" - Если валет оказывается картой-стартером, то сдатчику засчитывается очко.

Теперь, рассмотрим некоторые примеры подсчета очков.

Комбинация К, Д, Д, В.

К, Д, В - 3 очка.
К, Д, В - 3 очка.
Д, Д - 2 очка.
Итого: 8 очков

Комбинация К, Д, Д, В, 10:
К, Д, В, 10 - 4 очка
К, Д, В, 10 - 4 очка
Д, Д, - 2 очка
Итого: 10 очков

Комбинация К, Д, Д, Д, В:
К, Д, В - 3 очка,
К, Д, В - 3 очка.
К, Д, В - 3 очка.
Д, Д, Д, - 6 очков
Итого: 15 очков

Комбинация К, Д, Д, В, В:
К, Д, В - 3 очка.
К, Д, В - 3 очка.
К, Д, В - 3 очка.
К, Д, В - 3 очка.
Д, Д, - 2 очка.
В, В, - 2 очка.
Итого 16 очков.

Комбинация 9, 8,8,7,7:
8,7 - 2 очка
8,7 - 2 очка
8, 7 - 2 очка
8, 7 -2 очка
7,7 - 2 очка
8, 8 - 2 очка
9, 8, 7 - 3 очка
9, 8, 7 - 3 очка
9, 8, 7 - 3 очка
9, 8, 7 - 3 очка
Итого: 24 очка

Как видите, в крибедже не так просто правильно подсчитать все положенные Вам очки. В этом тоже есть интересный элемент игры. Считать нужно быстро и точно - этому вас научит правило МАГГИНЗ. Дело в том, что если Вы, объявляя свои очки, что-то забудете подсчитать, то Вам соперник имеет право Вас подправить, объявив "Маггинз!" и Ваши очки, которые он увидел, а Вы - нет, запишутся ему.

Вообще-то говоря, не принято при игре с новичками пользоваться этим правилом, поэтому в той программе, которая предлагается Вашему вниманию, есть возможность играть с этим правилом или без. Если Вы играете с ним, то сами должны ввести количество своих очков. Ошибетесь, компьютер сделает Вам "Маггинз!". Если Вы от этого правила откажетесь, то компьютер будет делать все расчеты и за Вас и за себя по всем правилам.

Стратегия в крибедже.

Исследование стратегий в крибедже можно разделить на два этапа. На первом этапе игроки должны определиться со сносом в криб. На первый взгляд, достаточно сосчитать все

очки в шести картах и отложить две с тем, чтобы сохранить максимальный счет в оставшихся четырех. Но иногда это приводит к тому, что приходится сносить в криб очки или ценные карты. Если криб принадлежит Вам, в этом нет ничего плохого, но если он принадлежит противнику, лучше ослабить руку и "надуть" криб. В криб противника очень опасно сносить пятерки, семерки, восьмерки и карты, которые могут стать основой для секанса. Лучшими для "надувания" являются очень высокие по достоинству, очень малые и карты "с разбросом", имеющие по достоинству промежуток в две и более карты.

Иногда приходится дробить комбинацию на руке, даже если криб принадлежит Вам. Как правило, лучше сохранить секанс и разбить пару, если это необходимо. Секанс в руке дает хорошие ожидания на поддержку от стартера, после того, как он будет открыт.

На втором этапе, в розыгрыше игры основной принцип стратегии состоит в том, чтобы не дать противнику объявить "пятнадцать" или "секанс". Самой надежной картой для первого хода является ход "четверкой", в ответ противник ни может сделать " пятнадцать", ни может помешать это сделать вам. Он, конечно, может дать "пару", но против пар защиты быть не может.

Десятиочковая карта (десятка или фигура) не очень хороший первый код, но он неплох, если у вас есть пятерка. Когда противник объявит "пятнадцать", Вы поставите "пару".

Гораздо приятнее ход, например семеркой. Если противник поставит "пару", Вы сможете тузом сделать "пятнадцать". Если противник поставит "пятнадцать" с помощью восьмерки, У Вас возможность дать "пару" и открытая и сверху и снизу секансная последовательность.

Одним словом, если у вас на руке есть "пятнадцать", например 9+6 то ходите старшей из них - это безопасно. При отсутствии других соображений ходить старшими более предпочтительно, сберегая младшие на конец раунда для игры под 31 очком.

Пример розыгрыша партии.

Рассмотрим в качестве примера розыгрыш одной партии, после чего перейдем непосредственно к нашей программе. В результате сдачи карт образовался следующий расклад:

СДАТЧИК:

ПК Б9 Т9 Б8 П7 ЧТ

ПРОТИВНИК

БВ П10 Ч8 Т7 Ч5 Т3

Противник сносит в криб 10-3.

Он не может сбросить 8-7 как прекрасную комбинацию (это "пятнадцать" и хорошая основа для "секанса"). Он не может сбросить 5, т.к. велика вероятность, что стартером окажется десятиочковая карта. Выбирая между бубновым валетом и пиковой десяткой, он предпочитает оставить валета, т.к. он может оказаться "его благородием".

У сдатчика уже есть двойной "секанс" 9-9-8-7, поэтому он легко сносит К-Т.

После переворота стартера оказалось, что это пиковая шестерка.

СТАРТЕР: П6

1. Противник заходит восьмеркой (можно было пойти и семеркой). - "Восемь".
2. Сдатчик делает ход семеркой. - "Пятнадцать" - "два очка".
3. Противник кладет семерку. - "Двадцать два" - "пара" - "два очка".
4. Сдатчик играет бубновой девяткой. - "Тридцать одно" - "два очка".
5. Противник открывает новый счет, он ходит валетом - "Десять".

6. Сдатчик может положить восьмерку или девятку, он кладет девятку, полагая, что девятки у противника нет, иначе тот на третьем ходу положил бы ее, делая "секанс", а он сделал "пару". - "Девятнадцать".

7. Противник играет пятеркой. - "Двадцать четыре".

8. У сдатчика восьмерка. Ее класть нельзя, т. к. сумма превысит 31 очко. Он объявляет "Ход!" и противник заносит себе очко.

9. У противника нет карт, счет открывается в третий раз. Ходит сдатчик, выкладывает свою восьмерку и получает очко "за последнюю карту".

Начинается подсчет комбинаций.

ПРОТИВНИК

"пятнадцать"	В-5	- 2 очка
"пятнадцать"	8-7	- 2 очка
"секанс"	8-7-6-5	- 4 очка
Итого		- 8 очков

СДАТЧИК

"пятнадцать"	8-7	- 2 очка
"пятнадцать"	9-6	- 2 очка
"пятнадцать"	9-6	- 2 очка
"пара"	9-9	- 2 очка
"секанс"	9-8-7-6	- 4 очка
"секанс"	9-8-7-6	- 4 очка
Итого		16 очков.

Сдатчик открывает криб, но там, к сожалению, нет ни одного очка.

Программа

Программа отличается значительным размером и представляет немалую сложность в отладке. Для упрощения этой трудоемкой работы мы привели в конце статьи комментарий к программе - он Вам поможет.

Программа была нами проверена уже после печати оригинал-макета и ошибки в нем крайне маловероятны, но опыт показал, что основную трудность представляет похожесть символов I, l и цифры 1. Поэтому в тех местах, где это особенно критично, мы даем пометку в строке REM.

На Ваше собственное усмотрение остается вопрос русификации Вашего компьютера. В вашем распоряжении память выше 60000, где вы можете разместить свой шрифт. О том, как это делается, читателям ZX-РЕВЮ, по-видимому, говорить не надо - мы об этом много раз писали. Можете этот адрес и изменить, скорректировав значение CLEAR в строке 9900.

Русификация с помощью символов UDG-графики здесь не проходит, т.к. они уже используются программой для изображения игральных карт и доски для крибеджа.

```
10 REM Здесь Вы введете команды,
20 REM необходимые для
30 REM русификации Вашего
40 REM компьютера
50 REM ************************
60 DEF FN t(x) = (x>9)*10+(x<10)*x
70 DEF FN s(x)=10*(x-INT (x))
80 GO TO 7000
500 REM ******************************** Выбор 2-х карт.
510 PRINT AT 7, 8; FLASH 1; "Я думаю"; AT 0,0;
565 FOR i=1 TO 5
570 FOR j=i+1 TO 6 : REM (I+1)
575 LET y=1: LET sum=0: LET f5=0
580 FOR x=1 TO 6
```

```
585 IF x=i OR x=j THEN GO TO 610
590 LET h(y)=c(x): LET i(y)=d(x) : LET h$(y)=c$(x): LET sum= sum+i(y)
595 IF i(y)=5 THEN LET f5=f5+1
600 LET y=y+1
610 NEXT x
620 GO SUB 1000
630 LET s=s+p+f+fl+f5: LET cr=0
640 IF c(j)=c(i) OR c(j)=c(i)+1 THEN LET cr=2
650 IF d(i)+d(j)=15 THEN LET cr=cr+2
652 IF debug THEN PRINT: FOR q=1 TO 4: PRINT h(q);" ";: NEXT q: PRINT , s;" ";cr;"   ";
655 IF dir=1 THEN LET s=s+cr: GO TO 665
660 LET s=s-cr
665 IF s>max THEN LET max=s: LET t(1)=i: LET t(2)=j: IF debug THEN PRINT "x";
670 NEXT j: NEXT i
675 LET y=0
680 LET x=t(1): GO SUB 5300
685 LET x=t(2): GO SUB 5300
690 PRINT AT 7,8;" ГОТОВ "
695 RETURN
1000 REM ********************************* Расчет руки
1070 LET p=0: LET f=0: LET fl=4 : REM (FL=4)
1110 IF sum=15 THEN LET f=2
1120 FOR x=1 TO c-1
1130 FOR y=x+1 TO c
1140 IF h(x)=h(y) THEN LET p=p+2
1150 IF i(x)+i(y)=15 THEN LET f=f+a
1155 IF c<5 THEN GO TO 1220
1160 LET t=0
1170 FOR z=1 TO c
1180 IF z=x OR z=y THEN GO TO 1200
1190 LET t=i(z)+t
1200 NEXT z
1210 IF t=15 THEN LET f=f+2
1220 NEXT y: NEXT x
1240 FOR x=1 TO x
1250 IF sum=1(x)=15 THEN LET f=f+2
1260 IF h$(x)<>h$(1) THEN LET f1=0: REM (FL=0)
1270 NEXT x
1360 IF c=4 THEN GO TO 1400
1290 IF sum-i(5)=15 THEN LET f=f+2
1300 IF fl=4 AND h$(5)=h$(1) THEN LET fl=5: REM (FL)
1310 IF fl=4 AND crib=2 THEN LET fl=0: REM Флешь в крибе может иметь не менее 5 карт.
1400 LET x=1: LET s=0
1410 IF x>3 THEN RETURN
1420 LET r=1: LET d=1
1430 IF h(x+1)=h(x)+1 THEN LET r=r+1: GO TO 1490
1440 IF h(x+1)>h(x) THEN GO TO 1470
1450 LET d=d+1: IF d<>3 THEN GO TO 1490
1455 IF h(x-1)<>h(x) THEN LET d=4
1460 GO TO 1490
1470 IF r>rlen THEN LET s=s+d*r : IF r=2 THEN LET s=s-d
1460 LET x=x+1: GO TO 1410
1490 LET x=x+1: IF x<c THEN GO TO 1430
1500 IF r>rlen THEN LET s=+d*r: IF r=2 THEN LET s=s-d
1510 RETURN
2000 REM ********************************* Выкладывание карт на стол
2005 LET p=0: LET s=0
2008 LET ct=FN t(c)
2010 IF sum+ct>31 THEN LET s=-1: RETURN
2055 LET n=n+1 : LET t(n)=c
2020 LET t=sum+ct: IF t=15 OR t=31 THEN LET s=2
2022 IF n=1 THEN RETURN
2025 IF ABS (t(n-1)-c)>= n THEN RETURN
2026 REM **Проверка на пару
2030 FOR x=n-1 TO 1 STEP 1
```

```
2040 IF t(n)<>l???(x) THEN GO TO 2100
2050 LET p=p+2
2060 NEXT x
2100 IF p=6 THEN LET p=12
2110 IF p=4 THEN LET p=6 2115 LET s=s+p
2120 IF p>0 THEN RETURN
2130 REM **Пар нет, проверим секансы
2200 IF n<3 THEN RETURN
2210 FOR I=3 TO n
2220 LFT y=1
2230 FOR x=n-I+1 TO n
2240 LET h(y)=t(x): LET У=У+1
2250 NEXT x
2270 GO SUB 2400
2330 LET r=l: REM (R=L)
2340 FOR x=1 TO I-1
2350 IF h(x)+1<>h(x+1) THEN GO TO 2360
2360 NEXT x
2370 IF r>p THEN LET p=r
2380 NEXT I
2385 LET s=s+p
2390 RETURN
2400 REM ******* Подпрограмма сортировки массива h() размерностью l;
2410 LET Z=0
2420 FOR x=1 TO l-1
2430 IF h(x)>h(x+1) THEN LET z=h(x): LET h(x)=h(x+1): LET h(x+1)=z
2440 NEXT X
2450 IF z<>0 THEN GO TO 2410
2460 RETURN
2700 REM ********* Ход игрока.
2710 IF nh = 4 THEN LET hgo=1: PRINT AT 6,9; "  ХОД!  ": RETURN
2715 LET m$= "Выберите карту": GO SUB 5500
2720 LET c=4: LET x=1: GO SUB 3700
2740 IF b(x)<>0 THEN GO TO 2765
2742 IF sum<22 THEN GO TO 2720
2745 PRINT AT 20, x*4-4; "ход?"
2747 LET m$= "Если нет хода, нажмите ЕНТЕР": GO SUB 5500
2750 LET hgo=x: GO SUB 3700
2755 PRINT AT 20, hgo*4-4; "    ": REM 4 пробела
2760 IF hgo<>x THEN GO TO 2740
2762 IF sum<safe THEN LET safe=sum
2763 RETURN
2765 LET k=x: BEEP .02,15
2770 LET c=a(k): GO SUB 2000
2775 IF s<0 THEN BEEP .2,20: LET m$="Сумма должна быть меньше 32": GO SUB 5500: GO TO 2720
2780 LET nh = nh+1
2785 LET sum=t: LET b(k)=0
2790 LET x = k: LET y=16
2800 GO SUB 5300
2820 LET x=nh: LET y=8
2830 LET x$=r$(c): LET y$=a$(k)
2840 GO SUB 5400
2650 LET player=man
2900 REM *** Подсчет очков
2910 PAPER 4
2920 PRINT AT 6,0; "Всего ";sum;"       ": REM 7 пробелов
2940 PRINT AT 6, 9; "   ";s; " "
2955 GO SUB 6000
2960 RETURN
3200 REM *** Ход компьютера
3220 IF nc =4 THEN LET cgo=1: RETURN
3225 LET m$="": GO SUB 5500
3230 IF debug=1 THEN PRINT #1; AT 0,0; n$; AT 0, 0;
3240 LET max=-9: LET x1=0
3250 FOR i=1 TO 4
```

```
3260 IF d(1)<0 THEN GO TO 3500: REM ход уже сделан
3270 LET c=c(i)
3280 GO SUB 2000: IF s<0 THEN GO TO 3500: REM не по правилам
3290 LET n=n-1: LET i(i)=s: REM Переиграть, счет сохранить
3300 REM специальные правила
3305 IF t+c=31 AND t<safe THEN LET s=s-1
3310 LET s=s+(t>15)-(t=21)+(t>=safe)-2*(t = 5)
3315 IF n>0 THEN GO TO 3400
3320 FOR j=1 TO 4
3330 IF i=j OR d(j)<0 THEN GO TO 3360
3340 IF t<>5 AND t+d(j)=15 THEN LET s=s+2
3350 IF ABS (c-c(j))<2 THEN LET s=s+2
3360 NEXT j
3380 GO TO 3450
3400 IF ABS (t(n)-c)>2 THEN GO TO 3450
3410 FOR j=1 TO 4
3420 IF j =i OR d(j) THEN GO TO 3440
3430 IF ABS (t(n)-c(j) )< = 2 THEN IF t+2*d(i)<32 THEN LET s = s+2
3440 NEXT j
3450 LET s=s+(RND)>.6)
3460 IF s>=max THEN LET max=s: LET x1=i
3490 IF debug THEN PRINT #1;c;"=":i(i);",";s;" ";
3500 NEXT i
3550 IF x1=0 THEN LET cgo=1: PRINT AT 6,9;" ХОД! ": RETURN
3560 LET c=c(x1): LET t=sum+FN t(c)
3570 LET n=n+1: LET t(n)=c
3580 LET sum=t: LET s=i(x1)
3590 LET nc=nc+1: LET d(x1)=-9
3600 LET x=nc: LET y=0
3610 LET x$=r$(c): LET y$=c$(x1)
3620 BEEP .02,12: GO SUB 5400
3630 LET player=zx
3640 GO TO 2900
3700 REM ********************************* Выбор карты
3710 PAPER 4
3720 PRINT AT 21,x*4-3; FLASH 1:
3725 IF INKEY$<>"" THEN GO TO 3725
3730 IF CODE INKEY$=13 THEN GO TO 3600
3750 IF INKEY$<>" " THEN GO TO 3730
3760 PRINT AT 21, x*4-3;" ";
3770 LET x=x+1: IF x>c THEN LET x=1
3790 GO TO 3720
3800 PRINT AT 21,x*4-3;" ";
3810 RETURN
4000 REM ********************************* Очередность ходов
4050 LET nh=0: LET nc=0
4065 LET safe=31
4080 GO SUB 4400
4090 IF dir<>zx THEN GO TO 4200
4100 REM ИГРОК
4110 IF done=1 THEN RETURN
4120 GO SUB 2700
4125 IF win>0 THEN RETURN
4130 IF sum=31 THEN GO SUB 4300: GO TO 4200
4140 IF cgo=0 THEN GO TO 4200
4150 IF hgo = 0 THEN GO TO 4100
4160 LET s=1: GO SUB 2900: IF win>0 THEN RETURN
4170 GO SUB 4300
4200 REM Компьютер
4205 IF done= 1 THEN RETURN
4210 GO SUB 3200
4220 IF win>0 THEN RETURN
4230 IF sum=31 THEN GO SUB 4300: GO TO 4100
4240 IF hgo=0 THEN GO TO 4100
4250 IF cgo=0 THEN GO TO 4200
```

```
4260 LET s=1: GO SUB 2900: IF win>0 THEN RETURN
4270 GO SUB 4300
4290 GO TO 4100
4300 REM ** Переворот карты
4310 LET y=0
4320 FOR x=1 TO nc
4325 GO SUB 5350
4330 NEXT x
4340 LET y=8
4350 FOR x=1 TO nh
4360 GO SUB 5350
4370 NEXT x
4400 REM **Следующий раунд
4405 LET done=0: LET s=0
4420 LET sum=0: LET n=0
4430 IF nh=4 AND nc=4 THEN LET done=1
4440 LET cgo=0: LET hgo=1
4450 GO SUB 2900: RETURN
4500 REM ********************************* Открытие и подсчет очков
4510 PRINT #1;AT 0,0;n$;
4515 LET =5: LET rlen=2
4520 LET x=18: GO SUB 5600
4530 IF dir=zx THEN GO TO 4600
4540 FOR x=1 TO 5
4550 LET h(x)=c(x): LET h$(x)=c$(x)
4560 NEXT x
4570 LET up=1: LET c=4: GO SUB 5100
4575 LET m$="Считаю свои очки":GO SUB 5500
4580 LET player=zx: GO SUB 4700
4585 IF win THEN RETURN
4590 IF dir=zx THEN GO TO 4660
4600 LET m$="Считаю Ваши очки":GO SUB 5500
4605 FOR x=1 TO 5
4610 LET h(x)=a(x): LET h$(x)=a$(x)
4620 NEXT x
4625 LET c=4: LET y=11
4630 GO SUB 5000
4640 LET player=man: GO SUB 4700
4650 IF dir=zx THEN GO TO 4540
4660 LET m$="Готовы вскрыть криб? ": GO SUB 5500
4561 PAUSE 0
4665 LET x=18: GO SUB 5600
4670 GO SUB 5200
4675 LET m$="Считаю очки в крибе": GO SUB 5500
4660 GO SUB 4700
4690 RETURN
4700 REM ********************************* Подсчет очков
4705 LET nob=0: LET sum=0
4710 FOR x=1 TO 5
4715 IF x<5 AND h(x)=1 AND h$(x)=e$(5) THEN LET nob=1
4720 LET i(x)=FN t(h(x))
4735 LET sum=sum+i(x)
4740 NEXT x
4750 LET l=5: GO SUB 2400
4760 LET c=5: GO SUB 1000
4765 PRINT
4770 LET m$="": GO SUB 5500
4775 IF player=zx OR NOT mug THEN GO TO 4820
4780 INPUT "Введите свой счет. " ;ss
4785 IF ss<0 OR ss>50 THEN GO TO 4780
4790 IF ss=s+p+f+fl+nob THEN LET m$="Я согласен": LET s=ss: GO TO 4880
4795 LET m$=STR$ ss+" - неверно. Я получаю очки!"
4800 LET mug=2: LET player=zx
4820 PAPER 4
4830 IF f>0 THEN PRINT "15-ть -";f;"  ";
```

```
4840 IF p>0 THEN PRINT "пары-";p;
4845 PRINT
4850 IF fl>0 THEN PRINT "Флешь-";fl;"  ";
4860 IF s>0 THEN PRINT "секанс-";s;
4865 PRINT
4870 IF nob=1 THEN PRINT "очко - его благородие"
4875 LET s=s+p+f+fl+nob
4880 PRINT "Всего = ";s;
4885 IF mug = 2 THEN PRINT " - МОИ!"
4890 GO SUB 5500: GO SUB 6000
4891 IF mug=2 THEN LET player=man: LET mug=l: PAUSE 100
4895 RETURN
4900 REM ********************************* Снятие колоды - стартер выкладывается пятой
      картой в руке
4905 LET y=8: GO SUB 5350
4910 LET m$="Теперь я снимаю колоду"
4915 IF player=man THEN GO SUB 5500: PAUSE 50: GO TO 4930
4920 LET m$="Снимите колоду - любой клавишей": GO SUB 5500
4925 IF INKEY$="" THEN GO TO 4925
4930 LET r=RND*40+12.5
4935 LET k=INT p(r)
4940 IF k=s THEN GO TO 4930
4945 LET k$=s$(FN s(p(r)))
4950 LET x$=r$(k): LET y$=k$
4955 LET x=x-.25: LET y=7
4960 GO SUB 5400
4965 LET a(5)=k: LET a$(5)=k$
4970 LET c(5)=k: LET c$(5)=k$
4975 LET e(5)=k; LET e$(5)=k$
4980 RETURN
4985 LET e(5)=k: LET e$(5)=k$
5000 REM выкладка вашей руки
5020 FOR x=1 TO c
5030 LET x$=r$(a(x)): LET y$=a$(x)
5040 GO SUB 5400
5050 NEXT x: RETURN
5100 REM выкладка руки компьютера
5110 LET y=0
5120 FOR x=1 TO c
5130 LET x$=r$(c(x)): LET y$=c$(x)
5150 GO SUB 5400
5170 NEXT x: RETURN
5200 REM выкладка криба
5210 LET crib=1: LET y=11: IF dir=zx THEN LET y=0
5220 FOR x=1 TO 5
5230 LET h(x)=e(x): LET h$(x)=e$(x)
5240 LET x$=r$(e(x)): LET y$=e$(x)
5250 IF x<5 THEN GO SUB 5400
5270 NEXT x: RETURN
5300 REM стирание карты
5310 PAPER 4: GO TO 5370
5350 REM закрытая карта
5360 PAPER 2
5370 LET x$=" ": LET y$=x$: GO TO 5420
5400 REM открытая карта
5410 PAPER 7: IF y$=s$(1) OR y$=s$(3) THEN INK 2
5420 LET x1=4*x-4
5440 PRINT AT y,xl;x$;"  ";AT y+1,xl;y$;"  "
5450 PRINT AT y+2,x1;"    ";AT y+3,xl;"  ";y$
5460 PRINT AT y+4,x1;"  ";x$
5470 PAPER 4: INK 0: RETURN
5500 REM печать сообщений
5510 PRINT #1;AT l,0;n$;AT 1,0; m$
5520 RETURN
5600 REM Очистка экрана
```

```
5630 PAPER 4: PRINT AT 0,0;
5650 FOR y=1 TO 22: PRINT TAB x: NEXT y
5690 PRINT AT 0,0: RETURN
5700 REM Изображение доски для крибеджа
5710 PRINT AT 0,25; PAPER 6; "ВЫСП" :REM 3 пробела
5715 PRINT TAB 25; PAPER 6;"       " : REM 7 пробелов
5720 FOR y=1 TO 6 5730 PRINT TAB 25; PAPER 6; " A A " : REM UDG-символы.
5740 PRINT TAB 25; PAPER 6;" A A " : REM UDG-символы.
5750 PRINT TAB 25; PAPER 6; " B B " : REM UDG-символы.
5760 NEXT y
5765 PRINT TAB 25; PAPER 6; "       " REM: 7 пробелов
5770 PRINT n$; PAPER 6
5775 PRINT AT 2,28;"E";AT 19,28;"E" : REM UDG-символы.
5780 LET m$="КРИБЕДЖ"
5790 FOR x=1 TO 7
5800 PRINT AT 2*x+1,28;m$(x)
5810 NEXT x
5820 PRINT AT 1,0;: PAPER 4: RETURN
6000 REM корректировка счета
6010 BEEP .2,10
6015 IF s=0 THEN RETURN: REM вход в демонстрационную подпрограмму
6020 LET ss=v(player)
6030 IF ss>0 THEN GO SUB 6400
6040 LET v(player)=s(player)
6050 LET s(player)=s(player)+s
6060 IF s(player)>120 THEN LET win=player: GO TO 6085
6070 LET ss=s(player)
6060 GO SUB 6400
6085 PRINT PAPER 6; AT 20,25;s(man)
6090 PRINT PAPER 6; AT 20,31-(s(zx)>99)-(s(zx)>9):s(zx)
6095 RETURN
6400 LET x=25: LET v=1
6405 IF player=zx THEN LET x=31: LET v=3
6410 IF ss>60 THEN LET ss=ss-60
6415 IF ss<31 THEN LET ss=31-ss: GO TO 6440
6430 LET ss=ss-30: LET x=27: LET v=3: IF player=zx THEN LET x=29: LET v=1
6440 LET y=1+(ss+INT ((ss-1)/5))/2
6450 IF y<>INT y THEN LET y=INT y+1: LET v=v+1
6460 PRINT OVER 1; PAPER 6; AT y,x;v$(v)
6490 RETURN
7000 REM ******************************** Инициализация
7005 RANDOMIZE: LET debug=0
7010 BORDER 4: PAPER 4: INK 0: CLS
7020 LET x=0: LET y=0: LET z=0: LET i=0: LET j=0
7025 DIM h(8): DIM h$(6): DIM i(6)
7030 DIM a(6): DIM a$(6): DIM b(6): REM рука игрока
7035 DIM c(6): DIM c$(6): DIM d(6): REM рука компьютера
7040 DIM e(6): DIM e$(6): REM рука криба
7050 DIM g(2): DIM s(2): DIM v(2): DIM v$(5): REM счет
7055 DIM p(52): DIM t (12): REM колода
7060 INPUT "Добро пожаловать!" "Вам нужны инструкции? "; i$
7070 IF i$(1):"y" OR i$(1)="Y" THEN GO SUB 9000: GO TO 7080
7075 PRINT #1; "Пожалуйста немного подождите. "
7080 FOR x=1 TO 13: READ k$
7085 FOR y=0 TO 7: READ z
7090 POKE USR k$+y,z
7095 NEXT y: NEXT x
7300 LET zx=1: LET man=2
7305 REM ***Графика пользователя
7310 LET v$="GFJIK"
7315 LET s$="HCDS"
7320 LET r$="T23456739RBДК": REM здесь R - символ UDG-графики.
7330 LET n$="                                    ": REM 32 пробела
7335 LET o$-"Нажмите любую клавишу"
7340 IF i$="y" THEN GO SUB 9050
```

```
7350 REM: подготовка колоды
7355 LET z=0
7360 FOR x=1 TO 13
7365 FOR y=. 1 TO .4 STEP .1
7370 LET z=z+l: LET p(z)=x+y
7375 NEXT y 7380 NEXT x
7390 LET g(man)=0: LET g(zx)=0
7400 REM ****************************** Начало новой игры
7404 LET mug=0
7405 INPUT "Играем с правилом МАГИН3? ";m$: IF m$="y" OR m$="Y" THEN LET mug=1
7406 GO SUB 5700
7410 PRINT #1;AT 0,0; "Младшая карта сдает. "
7420 LET s(zx)=0: LET s(man)=0: LET v(1)=0: LET v(2)=0
7425 LET dir=zx: LET s= 0: LET player=zx: LET win=0
7430 LET x=2: GO SUB 4900: LET s=k
7435 PAUSE 50
7440 LET player=man: LET x=5: GO SUB 4900
7450 IF s<k THEN LET dir=man
7470 PAUSE 50: LET x=23: GO SUB 5600
7500 REM ************************* Тасование колоды и сдача
7510 LET m$=" Я сдаю"
7520 IF dir=man THEN LET m$="Вы сдаете"
7530 PRINT #0;AT 0, 0; m$; n$
7535 LET m$= "Тасую колоду"
7540 GO SUB 5500
7610 FOR x=1 TO 51
7620 LET y=INT (RND*(53-x))+x: LET z=p(x); LET p(x)=p(y): LET p(y)=z
7630 NEXT x
7635 LET m$="": GO SUB 5500
7640 REM **Сортировка карт
7650 FOR y=1 TO 7 STEP 6
7655 LET z=0
7660 FOR x=y TO y+4
7670 IF p(x)>p(x+1) THEN LET z=x: LET j=p(x): LET p(x)=p(z+1):LET p(x+1)=j
7680 NEXT x: IF z>0 THEN GO TO 7655
7690 NEXT y
7700 REM ** сдача
7710 LET i=0: LET j=16
7720 IF dir=zx THEN LET i=16: LET j=0
7730 FOR x=1 TO 6
7740 LET y=1: GO SUB 5350
7750 LET c(x)=INT p(x): LET c$(x)=s$(FN s(p(x)))
7760 LET d(x)=FN t(c(x))
7770 LET y=j: GO SUB 5350
7780 LET a(x)=INT p(x+6): LET a$(x)=s*(FN s(p(x+6)))
7790 LET b(x)=1
7795 NEXT x
7799 REM******************************** Снос карт в криб
7800 LET c=6: LET y=16: GO SUB 5000
7805 IF debug THEN GO SUB 5100
7810 LET m$=" Подождите, пока я снесу 2 карты.": GO SUB 5500
7820 LET c=4: LET rlen=l: LET crib=0
7825 LET max=-99: LET t(1)=1: LET t(2)=6
7830 GO SUB 500
7840 LET m$="Выберите 2 карты": GO SUB 5500
7845 IF i$="y" THEN GO SUB 9300: GO SUB 9400
7850 LET c=6: LET y=16: LET x=1
7860 GO SUB 3700: GO SUB 5350
7870 LET t(3)=x: GO SUB 3700
7880 LET x$=r$(a(x)): LET y$=a$(x)
7885 IF x=t(3) THEN GO SUB 5350: GO TO 7860
7890 LET t(4)=x: GO SUB 5350
7895 REM** Закрытие рук
7900 FOR x=1 TO 2
7905 LET e(x)=c(t(x)): LET e$(x)=c$(t(x))
```

```
7910 LET c(t(x))=0
7915 LET e(x+2)=a(t(x+2)): LET e$(x+2)=a$(t(x+2))
7920 LET a(t(x+2))=0
7925 NEXT x
7930 LET y=1: LET z=1
7935 FOR x=1 TO 6
7940 IF c(x)=0 THEN GO TO 7955
7945 LET c(y)=c(x): LET d(y)=d(x)
7950 LET c$(y)=c$(x): LET y=y+1
7955 IF a(x)=0 THEN GO TO 7970
7960 LET a(z)=a(x): LET a$(z)=a$(x) : LET z=z+1
7970 NEXT x
7980 LET x=24: GO SUB 5600
7985 IF i$="y" THEN GO SUB 9450
7990 PRINT #0; AT 0,0; n$
7999 REM **** Ход картой
8000 LET s=0: LET player=dir
8010 LET x=6: GO SUB 4900
8020 IF k=11 THEN LET m$= "2 очка за его челядь": GO SUB 5500: LET s=2: GO SUB 6000: PAUSE
      40
8030 LET c=4
8040 LET y=16: GO SUB 5000
8050 GO SUB 4000
8060 IF win THEN GO TO 8300
8065 IF i$="y" THEN GO SUB 9200
8070 GO SUB 4500
8080 IF win THEN GO TO 8200
8100 LET m$="Готовы к следующему ходу ?": GO SUB 5500
8105 IF INKEY$="" THEN GO TO 8105
8110 LET dir = dir+1: IF dir>2 THEN LET dir=1
8120 LET x=25: GO SUB 5600
8130 IF i$<>"y" THEN GO TO 7500
8140 INPUT "Продолжать инструкции? " ; i$
8150 IF i$="Y" THEN LET i$="y"
8160 GO TO 7500
8200 REM ***Победитель
8210 LET m$= "Примите поздравления с победой!"
8220 IF win=zx THEN LET m$="Вам не повезло - моя победа!"
8230 PRINT #1;AT 0,0;m$
8240 PRINT PAPER 6; AT 18,28; FLASH 1;v$(5)
8250 FOR x=1 TO 60 STEP 2: BEEP .02,x: NEXT x
8260 LET g(player)=g(player)+1 8265 LET m$=o$: GO SUB 5500
8270 PAUSE 0
8280 LET x=23: GO SUB 5600
8290 PRINT AT 4,8; "Счет: ": AT 5,7
8300 PRINT AT 7,8; "Вы   ";g(man)
8310 PRINT AT 9,8; "Комп.";g(zx)
8320 INPUT "Сыграем еще? "; m$
8330 IF m$="n" OR m$="N" THEN STOP
8380 GO TO 7400
9000 REM ******************** Инструкции
9005 LET i$="y"
9010 CLS : PRINT "  Крибедж - игра для двух игро-
        ков. Каждый получает до 6
        карт, из которых должен 2
        снести, образуя третью руку,
        называемую крибом. Криб счи-
        тается в пользу сдатчика.
        Из колоды вскрывается карта,
        принадлежащая одновременно
        всем трем рукам - стартер."
9020 PRINT ' "    Тузы оцениваются в одно
        очко. Все фигуры - в 10 очков."
```

```
9030 PRINT : PRINT "   Во время игры Вы увидите, как
        за некоторые комбинации карт
        начисляются очки."
9040 RETURN
9060 LET m$=o$: GO SUB 5500
9065 PAUSE 0
9070 CLS: GO SUB 5700
9075 PRINT "    Полученные очки отмечаются
        на игровой доске перемещением
        колышков. Победит тот, кто
        первым наберет 121 очко (два
        жды обойдет доску)."
9090 LET m$="Для демонстрации нажмите пробел." : GO SUB 5500
9095 PAUSE 0: IF INKEY$<>" " THEN GO TO 9175
9100 LET s(man)=0: LET s(zx)=0
9105 LET m$="Посмотрим мои очки": GO SUB 5500
9110 FOR i=zx TO man
9115 LET player=i: LET win=0
9120 LET s(player)=0: LET v(player)=0
9130 LET s=INT (RND*10): GO SUB 6015
9140 IF win=0 THEN GO TO 9130
9145 PRINT PAPER 6; AT 18,28;v$(5)
9150 LET m$="Посмотрим Ваши очки":GO SUB 5500
9155 NEXT i
9160 LET m$ "Для повтора нажмите ПРОБЕЛ": GO SUB 5500: GO TO 9095
9180 PRINT AT 9,0;
      "Когда Вам надо выбрать " '
      "карту, перемещайте ука-" '
      "затель клавишей ПРОБЕЛ " '
      "и делайте свой выбор - " '
      "клавишей ENTER.        "
9190 PRINT AT 9,0;
      "Если Вы будете играть  " '
      "с правилом МАГГИН3, то " '
      "должны сами считать    " '
      "свои очки. Я отберу их," '
      "если Вы ошибетесь.     "
9195 RETURN
9200 LET x=25: GO SUB 5600
9210 PRINT
      "Каждый сам считает свои" '
      "очки                   "
9220 PRINT
      "Затем сдававший счита- " '
      ет криб.                "
9240 GO SUB 9305: GO SUB 9400
9245 LET m$=o$: GO SUB 5500
9250 PAUSE 0: LET x=25: GO SUB 5600
9255 LET x=6: LET y=8: GO SUB 5350
9260 LET x=x-.25: LET y=7: LET x$-r$(c(5)): LET y$=c$(5)
9270 GO SUB 5400
9280 RETURN
9300 PRINT AT 0,0;
9305 PRINT
      "Очки начисляется за    " '
      "комбинации:            " '
9310 PRINT "За 15 баллов......2 очка"
9320 PRINT "За пару...........2 очка"
9330 PRINT "За три...........6 очков"
9310 PRINT "За четыре.......12 очков"
9350 PRINT "За секанс..очко за карту"
9360 RETURN
9400 PRINT "Флешь из четырех..4 очка"
9410 PRINT "Флешь из пяти....5 очков"
9420 PRINT "Валет масти стартера...."
```

```
9430 PRINT ".................. 1 очко"
9440 RETURN
9450 PRINT AT 0,0;
      "Мы ходим по очереди,    " '
      "пока сумма очков не     " '
      "приблизится к 31.       "
9460 PRINT
      "Набравший 31 балл, по-  " '
      "лучает 2 очка. Ближай-  " '
      "ший к 31 получает очко."
9470 PRINT
      "Комбинации на руках да-" '
      "ют очки:                "
9510 GO SUB 9320
9520 PRINT
      "Если вам нечем ходить,  "
      "выбирайте пустую карту  "
9530 LET m$=o$: GO SUB 5500
9540 PAUSE 0: LET x=25: GO SUB 5600
9590 RETURN
9600 DATA "a"
9610 DATA 0,195,195,0,0,195,195,0
9620 DATA "b"
9630 DATA 0,195,195,0,0,0,0,0
9640 DATA "c"
9650 DATA 56,56,254,254,214,16,1 6,56
9660 DATA "d"
9670 DATA 16,56,124,254,254,124,56,16
9680 DATA "e"
9690 DATA 0,24,24,0,0,0,0,0
9700 DATA "f"
9710 DATA 240,255,240,0,0,0,0,0
9720 DATA "g"
9730 DATA 0,0,0,0,240,295,240,0
9740 DATA "h"
9750 DATA 66,238,254,124,124,56,16,16
9760 DATA "i"
9770 DATA 15,255,15,0,0,0,0,0
9760 DATA "j"
9790 DATA 0,0,0,0,15,255,15,0
9800 DATA "k"
9810 DATA 28,28,28,28,8,8,8,8
9820 DATA "s"
9B30 DATA 16,56,124,254,254,146,16,56
9840 DATA "r"
9850 DATA 0,76,210,82,82,82,76,0
9900 CLEAR 59999
9999 PAPER 7: BORDER 7: INK 0: CLS: LIST
```

Комментарий

Графика пользователя используется в строках: 5730, 5740, 5750, 5775,7310,7315 и 7320.

Стратегия работы программы состоит из двух фаз. Первая фаза - когда компьютер решает, какие же 2 карты ему следует снести в криб. Здесь программа просматривает все комбинации из шести карт по 4 и каждый раз рассчитывает силу руки. Оценивается также сила двух сносимых в криб карт. Этих комбинаций всего 15 и они занимают около 1 секунды. Всего расчет оптимального сноса занимает порядка 20 секунд. Это самая медленная часть программы, но если Вы попробуете с ней сыграть, то увидите, что это не намного дольше, чем ожидание решения от живого партнера, так что здесь со скоростью работы проблемы особой нет.

Вторая фаза - розыгрыш рук и ведение счета. Здесь программа рассчитывает итог,

который может быть получен при том или ином ходе и применяет некоторый эвристический подход, который в определенной степени может давать и непредсказуемые результаты. Те, кому интересно посмотреть, как это происходит, могут в строке 7005 задать для переменной debug значение LET debug=1.

Несмотря на то, что программа полностью написана на БЕЙСИКе, в ней многое сделано для структурирования. Наиболее часто использующиеся подпрограммы, вынесены в начало для минимизации времени доступа к ним. Ниже мы рассмотрим назначение основных подпрограмм.

Данные также структурированы.

Колода представлена массивом из пятидесяти двух чисел - p(). Целая часть каждого числа выражает собой достоинство карты (1...13), а дробная часть - ее масть (0.1...0.4). Массив организуется случайный образом во время тасования колоды.

Другие важнейшие массивы:
Партнер:
a()- достоинство карт;
a$()-масти карт;
b()-вспомогательный массив.

Компьютер:
c() - достоинство карт;
c$() - масти карт;
d()- вспомогательный массив.

Криб:
e()- достоинство карт;
e$()-масти карт;

Вычислительные массивы:
h()- достоинство карт:
h$()-масти карт;
i()-вспомогательный массив.
t()-массив, содержащий карты, выложенные на стол во время ходов. Карты стола.
s() -счет.

Все вычисления, связанные со счетом, производятся после копирования массивов того или иного игрока в вычислительные массивы. Использование прочих переменных мы рассмотрим в составе основных подпрограмм.

500. Подпрограмма рассматривает все возможные комбинации 4-х карт из шести с целью определения что же сносить в криб. Сами расчеты делает другая подпрограмма (1000), к которой происходит обращение по мере необходимости.

На входе в подпрограмму задаются массивы, описывающие руку компьютера - c(), c$(),d().

На выходе она выдает t(1) и t(2) - позиции тех карт, которые должны быть снесены в криб.

1000. Подпрограмма рассчитывает очки, имеющиеся на руке. Она вызывается не только при расчете сноса, но и при проведении расчетов после открытия карт. Параметр c указывает сколько карт принимаются в расчет. При расчете сноса c=4, при подсчете итогов c=5 (включая карту-стартер).

Другие переменные:
rlen - содержит минимальную длину секанса.
sum - суммарная сила карт руки.

h(),i(),h$() - временно содержат данные той руки, для которой идет подсчет очков.
crib=1, если расчет идет для криба (флаг криба).
На выходе эта подпрограмма выдает:
p - очки за "пары";
f - очки за комбинации "пятнадцать";
s - очки за секансы;
fl - очки за флеши.

2000. Подпрограмма переносит карту из массива игрока в массив стола (делается ход, если он законный). Изменяется сила руки игрока и общий счет. На входе подпрограмма получает:
c - номер карты, которой делается ход;
sum - текущая сила руки;
t() - массив карт уже лежащих на столе.
n - количество карт, лежащих на столе.
На выходе подпрограмма выдает:
s - текущий счет (если сделан был незаконный ход, то этот параметр выдается отрицательным);
t() и n - измененные значения.

2400. Подпрограмма выполняет сортировку поступившего массива h(), имеющего длину l (буква L).

2700. Подпрограмма обслуживает ход игрока.
Вход:
nh количество карт сыгранных с этой руки;
a(), a$ () - карты, имеющиеся на руке;
b() - содержит нули для карт, которые уже сыграны.
Выход:
hgo=1, если ни один ход не может быть сделан;
win=man, если игрок достиг 121 очка и победил.

2900. Подсчитывает текущую сумму очков, корректирует счет, если необходимо.
Вход:
sum - текущая сумма очков;
s - счет;
player - указывает на игрока.
player=zx - компьютер;
player=man - человек.
Выход: нет.

3200. Рассчитывает оптимальный ход с руки компьютера. Перебираются и оцениваются все возможные ходы.
Вход:
nc - количество карт сыгранных с этой руки;
c(), c$ - карты, имеющиеся на руке;
d () - содержит нули для карт, которые уже сыграны.
Выход:
cgo=1 , если ни один ход не может быть сделан;

3700. подпрограмма обеспечивает игроку выбор карты для хода путем перемещения указателя. SPACE – перемещение, ENTER - ход.
Вход:
c - количество карт на руке.

Выход:
x - номер избранной карты.

4000. Подпрограмма последовательно вызывает подпрограммы, отвечавшие за ход игрока и ход компьютера, Подпрограмма в строке 3400 используется для того, чтобы перевернуть карты на столе, когда текущий счет дойдет до 31. Вызов подпрограммы 4400 сбрасывает текущий счет на ноль.
Вход:
dir - содержит сведения о том, кто сдавал карты.
dir=zx - компьютер;
dir=man - человек.
Выход:
done=1, если ход сделав;
win - указывает на победителя;
win=zx - компьютер;
win=man - человек.

4500. Изображает на экране раскрытые карты игроков и результат подсчета очков.

4700. Печатает счет для руки. Если в силе правило "МАГГИНЗ", предлагает игроку самому ввести свой результат.
Вход:
mug=1, если действует "МАГГИНЗ";
h(),h$() - карты анализируемой руки.

4900. Подпрограмма выполняет "срезку" колоды для вскрытия стартера.
Вход:
x - позиция указателя в колоде.
Выход:
k - достоинство стартера;
k$ - его масть.

5000. Изображает на экране карты игрока.
y - вертикальная позиция на экране;
c - количество карт в руке.

5100. изображает на экране карты компьютера.
c - количество карт в руке.
up= 1, если карты следует положить в открытом виде.

5200. Изображает на экране карты криба и копирует их из e() в h() для подготовки к расчетам.

5300. Стирает карты с экрана.
x - позиция карты от 1 до 6.
y - позиция экрана по вертикали.

5350. Печатает карту, изображенную рубашкой вверх (в темную).
x - позиция карты от 1 до 6.
y - позиция экрана по вертикали.

5400. Печатает открытую карту.
x - позиция карты от 1 до 6
y - позиция экрана по вертикали.

x$ - достоинство карты;
y$ - масть.

5500. Печатает в нижней части экрана текст сообщения из переменной m$.

5600. Очищает левую часть экрана.

5700. Рисует доску для крибеджа.

6000. Корректирует счет на доске для крибеджа.

7000. Инициализация программы. Объявление массивов, задание графики пользователя, составление колоды.

7400. Начало игры. Сбрасывается счет, вытягиванием младшей карты определяется, кто будет сдавать первый.

7500. Тасуется и сдается колода.

7800. Сброс карт, вызывается подпрограмма 500 для определения карт сноса. Дважды вызывается подпрограмма 3700 для сброса карт игрока. Массивы рук перерабатываются для удаления образовавшихся зазоров.

8000. Начало розыгрыша раунда. Срезается колода вызовом 4900. Затем игроки по очереди ходят (4000) и карты открываются для подсчета очков (4500).

8200. Определяется победитель. Изменяется счет в геймах, начинается новый гейм.

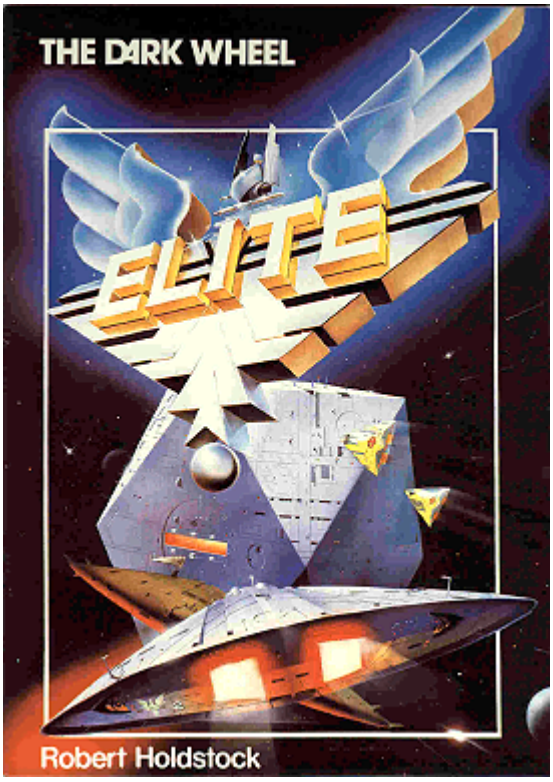
9000, Печать инструкции по игре. Переменная i$ определяет, давать инструкции или нет.

9600. Данные по UDG-графике.

9990. Инициализация цвета, установка верхней границы области БЕЙСИКа оператором CLEAR.

В заключение мы желаем Вам успеха в работе с этой интересной программой, надеемся, что у Вас достанет мужества и терпения, чтобы ее набрать и отладить. Ждем Вас в дальнейшем на страницах этой новой готовящейся книги.

Выполняя свое обещание, данное многочисленным поклонникам программы ELITE, сегодня мы начинаем печать повести THE DARK WHEEL, написанной Робертом Холдстоком по мотивам программы. Для нас это совершенно необычный опыт, т.к. в жанре художественного перевода мы, мягко говоря, себя никогда не пробовали. Что ж посмотрим, как будет выглядеть этот первый блин.



THE DARK WHEEL

ГЛАВА 1.

Торговый корабль "Авалония" плавно отошел от места орбитальной стоянки над планетой Лейв и начал маневр, приближаясь к исходной точке гиперперехода. Восемнадцать минут - ровно столько оставалось жить как кораблю, так и одному из двух членов ее экипажа.

Орбитальная станция отошла в тень, включились двигатели и, содрогаясь от вибрации, маленький корабль пошел к последнему гиперпрыжку. Внизу неспешно вращался Лейв во всем своем зелено-голубом великолепии. Шесть завитков пурпурных и белых облаков несли штормы морям, ливни континентам и обещали несколько влажных дней густым лесам и глубоким ущельям. Как яркие стеклянные осколки сквозь зеленоватое покрывало сверкали огни городов людей и лейвианцев.

Сидя за астронавигационной панелью, затаив дыхание, смотрел на этот пышный мир Алекс Райдер. Ему так и не разрешили сойти на поверхность планеты и этот вид вырвал из его груди вполне отчетливый вздох сожаления. Джейсон Райдер, его отец, сердито хмыкнул и заученным движением коснулся кнопок пульта управления. Кому-кому, а ему хорошо было известно то грустное чувство, которое испытывает астронавт при виде подобного великолепия с орбиты и не имея возможности прикоснуться к его роскоши. Он был однажды на поверхности Лейва. Незабываемое впечатление... Но правила и требования Галактической Кооперации Миров строги и разумны. Лейв, как и любая другая планета, не место отдыха и не объект для любопытства. Это живой, развивающийся мир и в нем живут люди, для которых он является тем же, чем когда-то была Земля для человечества: кровом, матерью, родным домом.

"В другой раз, в другое время" - решил Алекс. Он заслужит себе право посещений Лейва, а сейчас его карьера только начинается. Ему еще так долго учиться.

Райдеры были космическими торговцами в течение трех поколений. Основоположником дела был Бен Райдер. который торговал почти исключительно тем, что снимал с разбитых пиратских кораблей. Бен жил на грани и пришел день звездного года и настала ночь, когда он не вернулся. Ничто не нарушит уединения его могилы, сколь неизвестной, столь и далекой в межзвездной пустоте.

Его сын, а впоследствии и Джейсон Райдер, его внук, продолжили семейное дело. Скоро и Алекс примет главное решение - ставить на карту свою жизнь, снуя челноком между мирами или освоить другую профессию.

Космическая торговля! Давайте честно посмотрим правде в глаза. Это не развлечение для юнцов, одержимых идеей быстрого обогащения. Вы можете всю жизнь возить пищу, оборудование и текстиль и едва-едва наскрести крохи, чтобы купить клочок земли на побережье какой-нибудь планеты земного типа, чтобы провести остаток дней в тишине и уединенном комфорте.

Вот так.

Целая жизнь, пропитанная потом и кровью за дом и чистую голубизну чужого моря у его порога. Конечно, есть и другие пути, если Вы хотите большего: наркотики, рабы, экзотические животные, оружие, повстанцы, - займешься ими и богатство придет наверняка, а вместе с ним и пираты и рейдеры и каперы.

И полиция!!!

Усталость многих лет честной торговли уже сказалась на облике Джейсона, но он всегда по-немногу откладывал, и эта грузовая яхта была предметом ого радости и гордости. Он в любой момент мог прервать свою торговлю и ненадолго отдаться долгожданному отдыху, но справедливо полагал, что пустые трюмы бывают только у тех, у кого и в голове пусто, так что никогда не летал без груза, ради удовольствия. Вот и сегодня он был загружен соком экзотических ягод.

В этот рейс он взял с собой сына. Пусть парень посмотрит, что такое космос, может быть и у него появится интерес к семейному делу, пусть узнает, какова она жизнь в вечном вакууме.

Алекс Райдер был высоким светловолосым юношей. Он был отлично сложен и на своей планете, Онтиате, уже успел стать чемпионом по атмосерфингу. Как и любому другому парню, ему не терпелось поскорее перейти грань, отделяющую ученика от профессионала и начать строить стабильную жизнь: надежная девушка, надежная работа и первые планы по покупке земли.

Впрочем, у него еще был целый год для принятия решения, год серфинга, бейсбола в свободном падении, заоблачных пикников и прочих развлечений. Ему некуда спешить.

Но он еще любил космос, он любил этот солнечный блеск на обшивке корабля, этот грохот космопортов и неизведанность новых миров. Он обожал это чувство исследователя, первооткрывателя.

Голос из системы связи заставил его очнуться: "Авалония! До точки джамп-перехода четыре минуты малым ходом".

"Принято", - ответил Алекс и поднастроил автопилот, отец откинулся в кресле и ласково улыбнулся, ему пока нечего делать.

Опять голос системы управления: "Вход в джамп-переход по каналу два-семь, восток. Сорок пять".

"Принято", подтвердил Алекс и отец начал разворачивать корабль, готовясь к опасному гиперпереходу. " Всё выглядело спокойно.

На кормовом мониторе темная тень наползла на сверкающий диск планеты, - еще один корабль готовится к гиперпрыжку. Алекс не обратил на него особого внимания, сосредоточившись на предстоящем переходе. Отец придирчиво рассмотрел чужой корабль, но расслабился. Все в порядке. Как мог знать он, что жить ему оставалось всего четырнадцать минут.

Выполнить гиперпереход в такой сложной и перегруженной системе, как орбитальный

космопорт Лейва, дело очень непростое. Сотни глаз следят за каждым вашим движением, фиксируя малейшие просчеты. Одна ошибка в орбитальном маневрировании и в другой раз, на совсем другой планете, при подходе к станции "Кориолис" в космическом вакууме перед вами ярко вспыхнет транспарант "ПРИЕМА НЕТ".

Корабль дрейфует, строго подчиняясь указаниям службы мониторинга станции вместе с десятками других кораблей. Разворот - разгон – торможение - вращение, и все это с точностью до секунды: как по углу, так и по времени. Только так можно избежать столкновения хотя бы вон с тем двухтысячетонным торговцем, готовым протаранить двигательный отсек.

Далее управление берет на себя представительство службы Безопасности Полетов. Они проведут ваш корабль в толпе других торговцев, яхт, паромов, челноков, межзвездных лайнеров и стремительных полицейских патрулей. Серебряные стрелы прорезают темноту, ярко вспыхивают зеленые и голубые бортовые огни. Здесь и там мигают предупреждающие огни маяков и на мгновение экраны заслоняются стеной серого металла.

Вы продираетесь сквозь этот хаос и новый голос требует внимания - это служба Управления Навигацией Дальних Переходов. Она выведет вас в точку гиперпрыжка.

В течение нескольких минут вам предстоит преодолеть, скажем, семь световых лет и вы можете подумать, что это огромное пространство, но это не совсем так. Переход происходит по гипертуннелю, а это такой же туннель, как и любой другой. Он отличается лишь тем, что внутри него не существует обычного пространства - это магическое место, в котором обычные законы Вселенной не работают, туннель наводится под постоянным наблюдением и управлением. Здесь и там, через несколько тысяч парсеков, размещены спутники системы мониторинга, работают спасательные станции. Туннели сходятся и разветвляются и все их пространство пронизано сотней каналов, по которым идут корабли. Все сделано для защиты от двух главных опасностей гиперперехода: атомной реорганизации и смещения во времени.

Попробуйте своим ходом совершить гиперпереход хотя бы на пол световых года и вам очень повезет, если вы окажетесь в своей Вселенной.

Вы можете выйти из магического пространства вывернутый наизнанку (не очень приятное зрелище). Деформация пространства может не повредить вашему кораблю, но то желе, которое плавает в кабине - это вы. А еще ходят легенды о том, что полет может пройти и вполне благополучно и, спустившись с орбиты на Землю, вы будете долго недоумевать, чем вы помешали той гигантской ящерице и почему она так сильно переживает от вашего появления в ее славной доисторической пустыне.

Итак, в тот судьбоносный день Алекс Райдер внимательно прислушивался к механическим голосам роботов Службы Управления Навигацией, выводящим его в точку перехода к планете Листи. Он расслабился в кресле рядом с отцом и с интересом следил за работой космопорта, а за кормой нависла тень другого корабля, следующего к туннелю. То был грузовой корабль класса "Кобра".

Никто не знает, как так получилось, что космическим кораблям стали присваивать змеиные имена. Корабль Райдеров был довольно беззащитным "Офидионом", с двумя гипердрайвами и минимальным вооружением, пригодным для уничтожения разве что астероидов, метеоритов, и "сбесившихся кораблей". Так называли корабли неуправляемые или управляемые лихими юнцами ради потехи.

"Кобра" была куда более могучим кораблем.

"Кобры" - обычные торговые корабли, но в большинстве своем они погребены под горой оружия и всевозможнейших защит, которыми оснащают их крепко сбитые суровые капитаны. И на это есть причины…

Быть торговцем - это значит быть опасным и всегда рисковать. Опасным - потому, что если хочешь выжить, то должен хорошо знать свое оружие и как им пользоваться в космическом бою. Ты должен уметь мгновенно распознать пирата, анархиста или таргонский крейсер. Ты должен уметь обойти полицейские ловушки, если на борту есть хотя бы один из тысяч запрещенных видов товара. Рисковать - потому, что нет ничего слаще для корсара, чем жирная "Кобра", набитая мехами, минералами, рудами и текстилем. Быть

торговцем - это значит стрелять первым, а потом молиться, что ты не ошибся и твоя жертва действительно была пиратом. Ошибешься, и никакая броня и никакие ракеты не спасут тебя от "Вайперов".

"Вайперы"! Полицейские корабли. Маленькие, быстрые, смертоносные и невероятно цепкие. Пилот, конечно, человек, но убей человека и корабль продолжит атаку. Уничтожь корабль и будешь иметь дело с ракетой. Уничтожь ракету и всю оставшуюся жизнь шарахайся от каждой тени.

"Вайперы" не кусают. Они впиваются.

… Одиннадцать минут…

- Посмотри, не часто увидишь такое…

Слова отца прервали сосредоточенное изучение планеты, которым занимался Алекс. Справа параллельным курсом к гипертуннелю шел корабль странной формы, мигая мощными бортовыми огнями. Он сверкнул на солнце и Алекс увидел медленно вращающееся рыбье тело.

"Моури". Подводный корабль, способный летать в космосе. Его действительно редко можно было встретить в космосе. На таких планетах, как Регити и Аона, где только верхушки вулканов возвышаются над водой, "Моури" был и грузовым и пассажирским кораблем. Он был важнейшим средством связи с подводными городами.

Алекс с интересом рассматривал необычный корабль, а затем вновь обратился к кормовому экрану.

- Просигналить ей, чтобы держалась подальше?

Джейсон покачал головой, только сейчас Алекс понял, что и отец уже давно следит за этим кораблем. На мостике "Авалонии" возникло напряженное ожидание, это было непривычно и это было неприятно.

Что-то было не так. Алекс не знал, что именно, но все сильнее чувствовал это.

Что-то шло не по раз и навсегда установленному распорядку.

Вспыхнул сигнал, разрешающий вход в гипертуннель, раздался сопровождающий звуковой сигнал, в этот момент жизнь "Авалонии" съежилась до девяти минут.

Вблизи входа в гипертуннель всегда роятся стаи транзитных кораблей. Большинство из них швартуются группами к орбитальным буям. Механики и ремонтники используют эти часы вынужденного простоя для того, чтобы еще раз проверить и подремонтировать внешнее оборудование, в таком месте, на такой перегруженной системе, как Лейв, можно увидеть корабли любого когда-либо выпускавшегося типа.

По мере приближения к туннелю Алекс практиковался в распознавании кораблей - весьма необходимый навык для космического торговца. Распознать не пилотируемые орбитальные челноки было довольно легко, он увидел два "Эспа", принадлежащих военному флоту - небольшие высокоманевренные, смертоносные корабли, прекрасно защищенные от ударов, оснащенные самыми современными боевыми системами. Еще он увидел "Крейт", так называемый "старстрайкер" - маленький одноместный корабль, очень любимый первопроходцами и торговцами.

Справа стыковалась для высадки пассажиров цилиндрическая масса "Анаконды" - массивный грузовик, переделанный под пассажирские перевозки. Это был безобразно некрасивый корабль с распахнутыми приемниками космической пыли на носу.

Можно было составлять каталог. Вот "Боа" крейсерского класса, это "Питоны", а вот мечта охотников за призами "Фер-де-Ленс" - плотно упакованный оружием роскошный дворец.

Большие и малые, "Уормы", "Сайдуиндеры", "Мамбы" … - все это сверкало, мигало, отражало солнечные лучи своими серо-голубыми телами.

Как всегда, здесь были и рекламные корабли-роботы, предлагающие все, что угодно от "настоящего земного эля с медом фирмы Роганз" до "последнего обеда перед входом в гипертуннель".

- Пошли… Пристегнись…

Джейсон всегда говорил так. Алекс напрягся, хотя на самом деле вход в магическое пространство происходит с ничтожной перегрузкой. Мгновенное чувство головокружения, и

вот перед вами неописуемая картина звезд, разлетающихся многоцветными концентрическими кольцами. Впечатление, как будто звездолет летит сквозь вращающуюся трубу, мгновение, и все кончено. Корабль дрейфует в магическом пространстве, где нет ни места, ни времени. Он пересекает огромные пространства между мирами, а сам в эти секунды находится в мире, который нельзя ни представить, ни описать.

Говорят, что в магическом пространстве есть привидения. Может быть поэтому его и зовут магическим. Время сворачивается, а атомы выворачиваются, гравиволны громоздятся друг на друга. Что-то там движется, что это? Живые формы или тени? Атомы или галактики? - Кто знает, нельзя остановиться и выйти наружу посмотреть, здесь могут работать только дистанционные роботы, управляющие посты, распределительные станции и спасательная автоматика, то, что живет в магическом пространстве навеки останется тайной для людей.

Но привидения там точно есть. По крайней мере, призраки первых кораблей, что вошли в гиперпереход и никогда не вернулись.

Да, привидения… и еще тени… Змеиные тени… "Кобра" нависла над ними…

- Боже, что это…?

Джейсон Райдер стал белее снега.

В этом гипертуннеле он ничего не мог сделать, уклоняясь от другого корабля. Алекс воскликнул.

- Ведь он не знает правил! Может, это новичок?

- Возможно, ответил отец, не отрывая глаз от экранов радаров.

С покрытым испариной лицом Алекс следил за надвигающейся тенью "Кобры".

"Прекрасное оснащение… топливоприемники, контейнеры ракет, дополнительные грузовые пилоны, плоский купол отсека энергетической бомбы… богатый корабль… и смертоносный… ", - мысли неслись стремительно.

- Они же не собираются на нас напасть.

- Черта-с-два они не собираются.

… Три минуты …

(Продолжение следует)

Оглавление